# Toolbox PLUS+™

For CP-30 and G30 RTUs

**Version 3.11.0**

KINGFISHER

# Document Information

Copyright © 2007-2013 CSE Semaphore (Australia) Pty Ltd. ABN 35 006 805 910

Web: http://www.cse-semaphore.com

Email: info.kingfisher@cse-semaphore.com

*Kingfisher, Kingfisher PLUS*+ and *Toolbox PLUS*+ are trademarks of CSE Semaphore. *ISaGRAF* is a trademark of ICS Triplex ISaGRAF Inc. All other product names are trademarks of their respective owners.

Document Rev: 46

This manual describes **Toolbox PLUS+ Version 3.11.0**

This version of Toolbox PLUS+ should be used with Kingfisher CP-30 and G-30 based RTUs running **firmware Version 2874** or later.

It may also be used with RTUs with earlier firmware, provided that you only work with existing projects. Newly created projects may contain features which are not supported by the older firmware.

# Contents

1. Introduction ..... 1
1.1 About this Manual ..... 1
2. Software Installation ..... 3
2.1 Full install ..... 3
2.2 Software Updates ..... 4
3. Navigation ..... 5
3.1 Software Layout ..... 5
3.2 Menu Bar ..... 6
3.3 Multiple ways to select a menu ..... 9
3.4 Workspace ..... 10
4. RTU Configuration ..... 12
4.1 Overview ..... 12
4.2 Projects, Groups and RTUs ..... 12
4.3 Project Properties ..... 14
4.4 Add Modules ..... 15
4.5 RTU Properties ..... 17
4.6 RTU Properties – Protocols ..... 18
4.7 RTU Properties – Ports ..... 30
4.8 RTU Properties – Routes ..... 38
4.9 RTU Properties – Phone ..... 45
4.10 RTU Properties – Programs ..... 46
4.11 RTU Properties – Redundancy ..... 48
4.12 Module Properties ..... 49
5. Dictionary ..... 58
5.1 Dictionary Variables ..... 58
5.2 New Dictionary Variables ..... 60
5.3 Editing Variables ..... 63
5.4 Exporting and Importing Variables ..... 65
6. Maps ..... 66
6.1 Viewing RTU Locations ..... 66
6.2 Positioning an RTU ..... 67
6.3 Finding an RTU ..... 67
7. ISaGRAF ..... 68
7.1 ISaGRAF Overview ..... 68
7.2 Function Blocks ..... 69
7.3 Getting Started ..... 69

7.4 How Logic is Executed ..... 78
7.5 ISaGRAF Tips ..... 80
7.6 ISaGRAF Variable Types ..... 81
7.7 ISaGRAF Constants ..... 84
7.8 ISAGRAF Reserved Names ..... 85
8. ISaGRAF Function Blocks ..... 87
8.1 Kingfisher Protocol ..... 90
8.2 DNP3 Protocol ..... 98
8.3 Modbus Protocol ..... 106
8.4 Allen Bradley DF1 Protocol ..... 108
8.5 HART Protocol ..... 110
8.6 SNMP Client Protocol ..... 112
8.7 SNMP Trap Protocol ..... 116
8.8 SNMP RMS Trap Protocol ..... 118
8.9 User Defined Protocol ..... 120
8.10 VRRP Protocol ..... 121
8.11 General Communications ..... 122
8.12 Event Logging ..... 128
8.13 RTU System Data ..... 130
8.14 Maths and Logic ..... 135
8.15 PID Controller ..... 140
8.16 Gas Flow Calculations ..... 142
8.17 Obsolete Function Blocks ..... 153
9. ISaGRAF - Logic Examples ..... 154
9.1 Detecting Modules ..... 155
9.2 Scaling ..... 156
9.3 Hours ON ..... 157
9.4 Counting Pulses And Starts ..... 158
9.5 Flow Totalisation ..... 159
9.6 Daily Totals ..... 160
9.7 Exception Reporting Digitals ..... 161
9.8 Exception Reporting Analog Variables ..... 163
9.9 Event Logging ..... 164
9.10 Logic Examples – Polling ..... 166
9.11 Modbus Protocol ..... 170
9.12 Modbus Extended Addressing ..... 173
9.13 Allen Bradley Protocol ..... 174
10. Redundancy ..... 176
10.1 Redundant Processors ..... 176

10.2 Redundant Power Supplies ..... 180
10.3 Redundant Communications ..... 180
10.4 Redundant PCs ..... 181
11. Download ..... 183
11.1 Overview ..... 183
11.2 Firmware Compatibility ..... 183
11.3 RTU Restart ..... 184
11.4 Connecting to an RTU ..... 184
11.5 Downloading Configurations ..... 189
11.6 Downloading Firmware ..... 191
12. Viewing Data ..... 194
12.1 Status ..... 194
12.2 Event Logs ..... 197
12.3 Communications ..... 200
13. Appendices ..... 205
13.1 Glossary ..... 205
13.2 Kingfisher PLUS+ Specifications ..... 206
13.3 Kingfisher PLUS+ General ..... 207
13.4 Kingfisher PLUS+ Protocol ..... 208
13.5 Creating Variables Using Excel ..... 210
13.6 RTU Variables ..... 217

# 1. Introduction

A Remote Telemetry Unit (RTU) is a device that contains processing and communications equipment and is often located in a remote place. Kingfisher RTUs can interface to switches, relays and sensors, and connect to other intelligent devices via a wide range of supported protocols.

Kingfisher RTU features include:

- A wide range of modular analog and digital I/O
- Non-volatile event logging
- Support for diverse communications media – data radios, dialup and cellular modems, leased line, Ethernet and more…
- PLC-like logic processing utilizing all IEC 61131-3 languages
- Live logic debugging
- Support for multiple protocols in the same installation: Kingfisher, Modbus, DNP3 (with secure authentication), SNMP, Allen Bradley DF1 and more…
- Support for redundant power supplies, redundant processors and redundant communications – to maximise system availability

This manual describes **Toolbox PLUS+**, which is a Windows based software application used to configure and monitor "series 3" Kingfisher RTUs. This includes modular Kingfisher PLUS+ systems using the CP-30 processor, and the G30 standalone RTU.

The manual also covers the basic usage of **ISaGRAF**™, which is used for logic entry and debugging. Further details may be found in the ISaGRAF on-line help.

Note: For Kingfisher modular RTUs based on the earlier CP-12 or PC-1 processors, the G3 standalone RTU, and the LP-3 low power RTU, **Toolbox 32** software is used. This manual does not cover this software.

## 1.1 About this Manual

This manual is organised as follows:

- Section 2, Software Installation describes how to install Toolbox PLUS+, ISaGRAF and other software components;
- Section 3, Navigation discusses the basic layout of the application, and how to navigate the user interface.
- Section 4, RTU Configuration describes how to create a Toolbox PLUS+ project and define one or more RTUs. All RTU, port, module, route and protocol settings are explained.
- Section 5, Dictionary describes the facilities for managing the list of variables, or Dictionary, that is maintained by Toolbox PLUS+ and ISaGRAF.
- Section 6, Maps discusses how to indicate the position of an RTU on a map.

- Section 7, ISaGRAF is an introduction to ISaGRAF, which is used for entering logic to be executed by the RTU.
- Section 8, ISaGRAF Function Blocks is a reference for all custom function blocks supplied with Toolbox PLUS+. Refer to the ISaGRAF on-line help for standard built-in ISaGRAF function blocks.
- Section 9, ISaGRAF - Logic Examples includes some practical logic fragments for performing common operations.
- Section 10, Redundancy discusses how to maximise your system's availability by taking advantage of the RTU's support for redundant processors, power supplies and communications.
- Section 11, Download describes the procedures for updating the RTU's stored configuration settings, logic and firmware.
- Section 12, Viewing Data describes how to use Toolbox PLUS+ to manage, troubleshoot and retrieve data from a running RTU system.
- Section 13, Appendices, contains useful reference data, including:
  - Glossary of terms
  - RTU Specifications
  - Editing variables using an external spreadsheet
  - Automatically defined variables for each module type

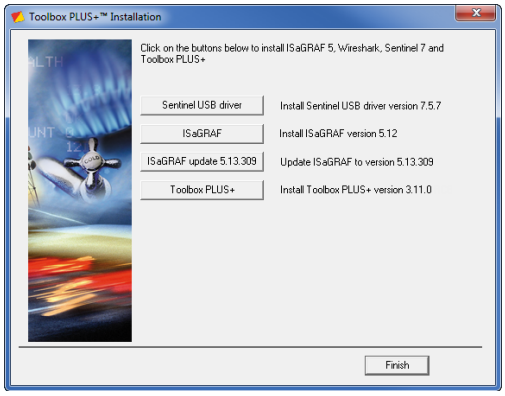
# 2. Software Installation

## 2.1 Full install

Toolbox PLUS+, ISaGRAF Workbench, and various other files and utilities are normally supplied on a USB flash drive. If you purchased an ISaGRAF license then you should also have received a USB license key “dongle”. (ISaGRAF can be used in trial mode for up to 30 days without the license key.)

It is recommended that you close all other programs before starting the installation. You may also need to temporarily disable anti-virus programs.

During installation, you may be prompted to restart your computer. This can be delayed until after all applications have been installed.


Toolbox PLUS+™ Installation
Click on the buttons below to install ISaGRAF 5, Wireshark, Sentinel 7 and Toolbox PLUS+
Sentinel USB driver
Install Sentinel USB driver version 7.5.7
ISaGRAF
Install ISaGRAF version 5.12
ISaGRAF update 5.13.309
Update ISaGRAF to version 5.13.309
Toolbox PLUS+
Install Toolbox PLUS+ version 3.11.0
Finish

Insert the Toolbox PLUS+ USB flash drive into the PC.

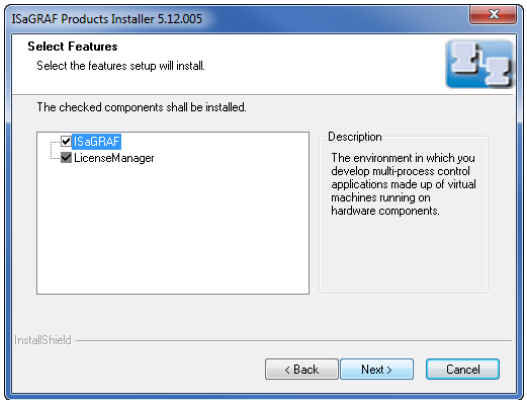
Open the drive in Windows Explorer and double click on the file: **autorun.exe**

The installation menu should be displayed.

Sentinel System Driver Installer 7.5.7 - InstallShield Wizard
Welcome to the InstallShield Wizard for Sentinel System Driver Installer 7.5.7
The InstallShield(R) Wizard will allow you to modify, repair, or remove Sentinel System Driver Installer 7.5.7. To continue, click Next.
Sentinel
< Back
Next >
Cancel

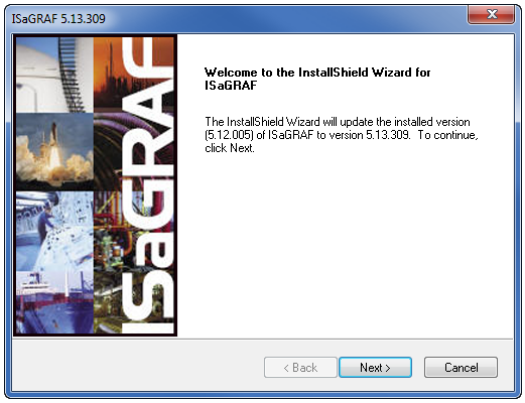
Click the **Sentinel USB Driver** button on the installation menu and follow the prompts to install it on your computer.

This driver provides support for the ISaGRAF USB protection key.

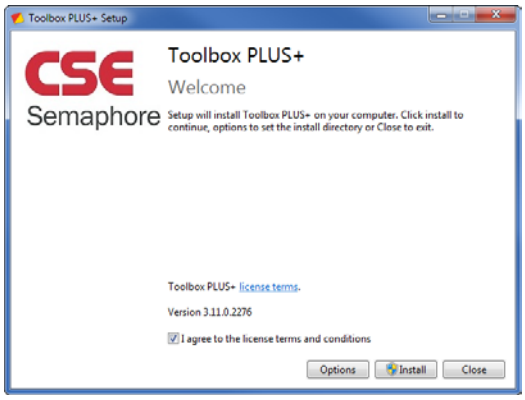

ISaGRAF Products Installer 5.12.005
Select Features
Select the features setup will install.
The checked components shall be installed.
ISaGRAF
LicenseManager
Description
The environment in which you develop multi-process control applications made up of virtual machines running on hardware components.
InstallShield
< Back
Next >
Cancel

Click the **ISaGRAF** button on the installation menu and follow the prompts to install it on your computer.

ISaGRAF allows you to develop logic programs to run on the RTU.


ISaGRAF 5.13.309
Welcome to the InstallShield Wizard for ISaGRAF
The InstallShield Wizard will update the installed version (5.12.005) of ISaGRAF to version 5.13.309. To continue, click Next.
< Back
Next >
Cancel

Click the **ISaGRAF 5.13.309 update** button in the installation menu and follow the prompts to install it on your computer.


Toolbox PLUS+ Setup
CSE Semaphore
Toolbox PLUS+
Welcome
Setup will install Toolbox PLUS+ on your computer. Click install to continue, options to set the install directory or Close to exit.
Toolbox PLUS+ license terms.
Version 3.11.0.2276
I agree to the license terms and conditions
Options
Install
Close

Click the **Toolbox PLUS+** button in the installation menu and follow the prompts to install it on your computer.

Installation is now complete. If you were requested to restart your computer, do so now.

If you have an ISaGRAF USB protection key, insert it into a USB port on your computer.

You can now try out Toolbox PLUS+!

## 2.2 Software Updates

The latest version of Toolbox PLUS+ can be downloaded from the CSE-Semaphore support website (http://helpdesk.cse-semaphore.com/). It is recommended that you keep your copy of Toolbox PLUS+ up to date by downloading and installing updates as they become available.

Note that the download package contains Toolbox PLUS+ software only; you will need to already have installed ISaGRAF 5.13.309 from a Toolbox PLUS+ USB flash drive or CD.

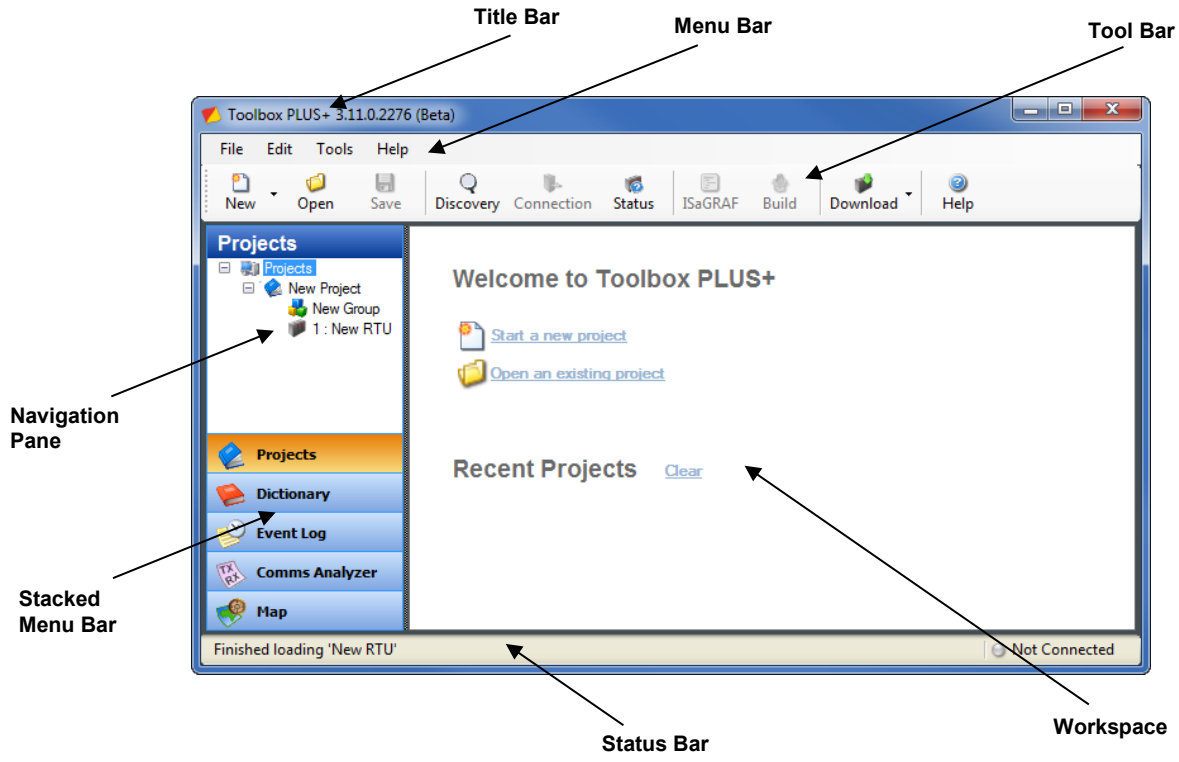
To install an update, simply double click on the downloaded executable file. This will start the Toolbox PLUS+ installer.

**Release notes** are available on the website and on the installation media. These describe the changes made in each version, and any other special instructions that may be required. Be sure to read these before installing or using Toolbox PLUS+.

# 3. Navigation

## 3.1 Software Layout

Toolbox PLUS+ employs a layout similar to many Windows applications. The Workspace and many of the menus are contextual meaning the display will vary according to what is currently selected.




**Title Bar:** Displays the name of the active project (if open) and the Toolbox PLUS+ program version

**Menu Bar & Tool Bar:** Allows access to all Toolbox PLUS+ commands. Note: some commands are only available when an RTU is selected in the navigation pane

**Status Bar:** Displays information about the progress or result of an action.

**Stacked Menu Bar:** Selects between broad categories of information to be displayed in the Workspace:

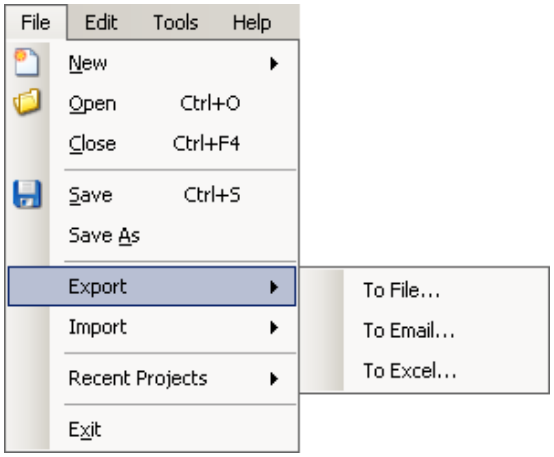
**Navigation Pane:** Displays the hierarchy of projects, groups and RTUs. Clicking on an item displays relevant information in the Workspace; double clicking allows the item's properties to be viewed and edited.

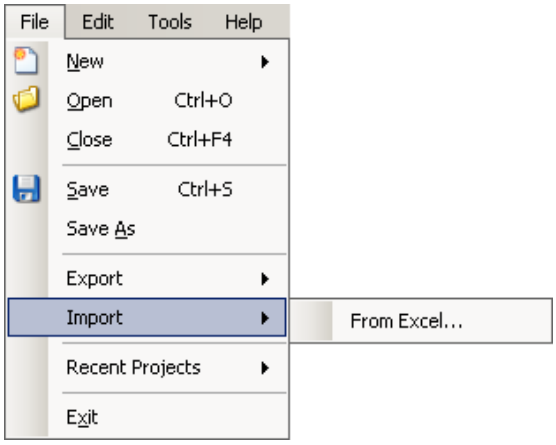
**Workspace:** Displays variable information depending on the currently selected items in the navigation pane and stacked menu bar. This may include system configuration, modules, variables and event logs

## 3.2 Menu Bar

The commands available from the Menu bar options – *File*, *Edit*, *Tools* and *Help* are described below. Some commands can be accessed using shortcut keys. These are listed alongside the Menu Bar commands shown below.

### 3.2.1 File Menu




**New:** Allows a new project, group, RTU, module or variable/s to be created.

Note the menu items are disabled if not applicable to the current selection.

**Open:** Opens an existing project.

**Close:** Closes selected project

**Save:** Saves selected project

**Save As:** Allows project to be saved with a new name and in a new location.

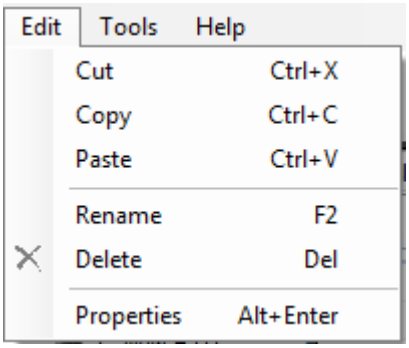
**Export:** Allows an entire project to be migrated to another computer using the *To File* or *To Email* option, or the projects dictionary and symbol information to be exported to Excel for editing.

**Import:** Allows dictionary variables to be imported from an Excel spreadsheet. The format of the spreadsheet must be the same as the spreadsheet created using the Export To Excel function above.

**Recent Projects:** Recently opened projects are listed here.

**Exit:** Closes Toolbox PLUS+ program.

### 3.2.2 Edit Menu




**Cut:** Copies the selected RTU configuration and then deletes it from the project.

**Copy:** Copies the selected RTU configuration.

**Paste:** Pastes the last copied RTU configuration into the selected group or project.

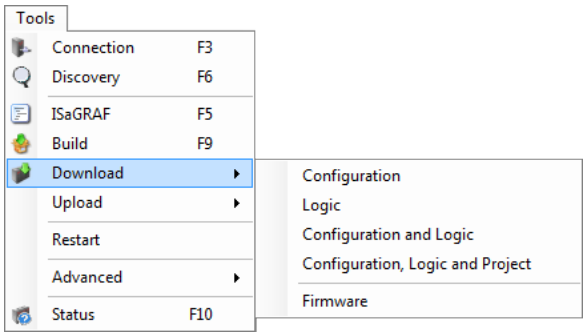
**Rename:** Allows the name of a project, group or RTU to be changed. Names can include spaces, hyphens ( - ), underscores ( _ ),

commas ( , ) and periods ( . ).

**Delete:** Deletes the selected group, RTU or module (modules are selected in the workspace). Note projects cannot be deleted, only closed. If you wish to delete a project, Windows Explorer is required.

**Properties:** Allows the settings for a project, group, RTU or module to be changed.

### 3.2.3 Tools Menu




**Connection:** Contains settings for how to communicate with the selected RTU. Only available when a project, group or RTU is selected. See Connection Parameters.

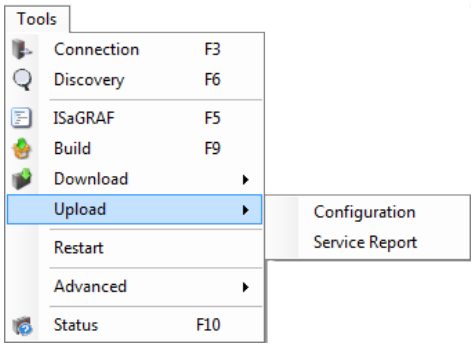
**Discovery:** Detects all RTUs on the same subnet as the PC Ethernet port(s), or connected to the PC's serial port.

**ISaGRAF:** Launches the ISaGRAF logic editor for the selected RTU.

**Build:** compiles the configuration and ISaGRAF programs for the selected RTU.

**Download:** Downloads the compiled Configuration and/or Logic (and optionally the complete project for later retrieval) to the selected RTU. Toolbox PLUS+ will automatically compile the configuration and/or logic if required before downloading. See Download for more details.

**Download > Firmware:** Downloads new CP-30, MC-31 or G30 firmware. See Download for more details.

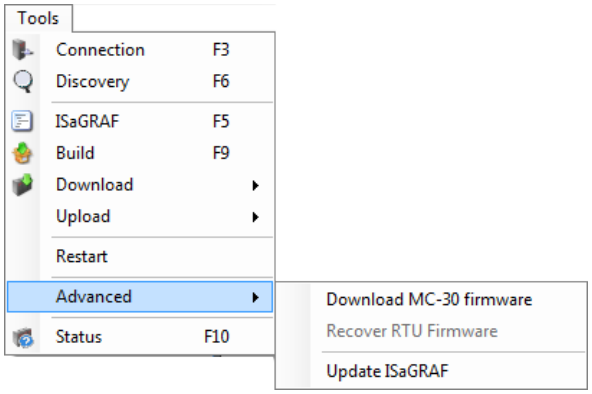



**Upload > Configuration:** Retrieves the current configuration from the connected RTU and creates a new project from it. This project will contain module definitions and settings, but not any user logic running on the RTU, unless you chose the *Configuration, Logic and Project* option when the configuration was downloaded to the RTU.

**Upload > Service Report:** Requests the RTU to prepare a diagnostic service report (requires firmware version 2918 or later). If you encounter a problem with the RTU, our support team may ask you to email the service report that the problem can be

diagnosed more quickly.

**Restart:** Restarts the selected RTU.




**Advanced > Download MC-30 firmware:** Used to download firmware to an MC-30 module.

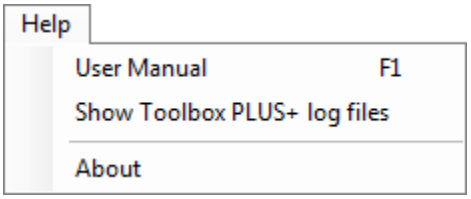
**Advanced -> Update ISaGRAF:** Updates an ISaGRAF project database which was created with a previous version of Toolbox PLUS+. This is not essential, but it will make available any new ISaGRAF function blocks which have been introduced in the intervening time.

If an RTU is selected then the selected RTU's ISaGRAF project database will be updated. If no RTU is selected then the databases for all RTUs in the project will be updated.

Caution! Please ensure the RTU has firmware that supports the new function blocks, to avoid possible RTU malfunction.

**Status:** Displays live status information for the selected RTU. Multiple module status windows can be viewed at the same time for the one RTU.

### 3.2.4 Help Menu



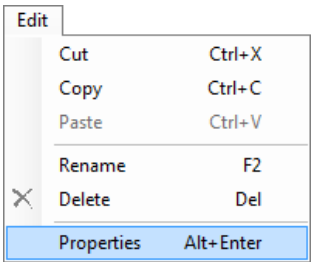

**User Manual:** Displays the Toolbox PLUS+ User Manual.

**Show Toolbox PLUS+ log files:** This will open the folder where Toolbox PLUS+ stores diagnostic logs during operation. If you encounter a problem using Toolbox PLUS+, our support team may ask you to email the files in this folder so that the problem can be diagnosed more quickly.

**About:** Displays Toolbox PLUS+ version number and other information.

## 3.3 Multiple ways to select a menu

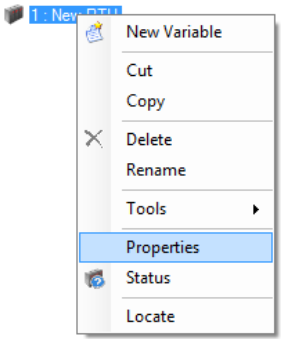
In many cases there are multiple ways to select commonly used functions. The examples below show different ways to edit the RTU properties.




**Via Menu Bar:** Select the RTU name in the navigation pane.

Then select *Edit* → *Properties*.

**or**




**Via Right-click menu:** Right-click the RTU name in navigation pane.

Then select *Properties*.

**or**

**Via Double-click:** Double-click the RTU name in the navigation page

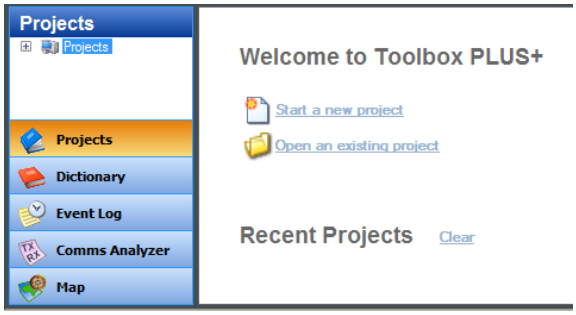
**Via Keyboard Shortcut:** Select the RTU name in the navigation pane and press Alt+Enter.

## 3.4 Workspace

The workspace area is context-specific, meaning that it will change according to what is currently selected in the navigation pane and stacked menu bar.

The various Workspace displays are shown below:

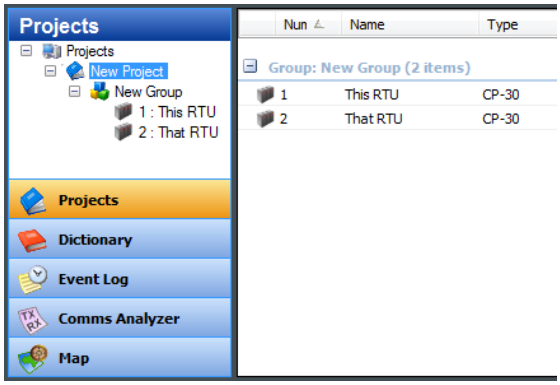
### Default View




Displayed when Toolbox PLUS+ is first started or Projects is selected in the navigation pane.

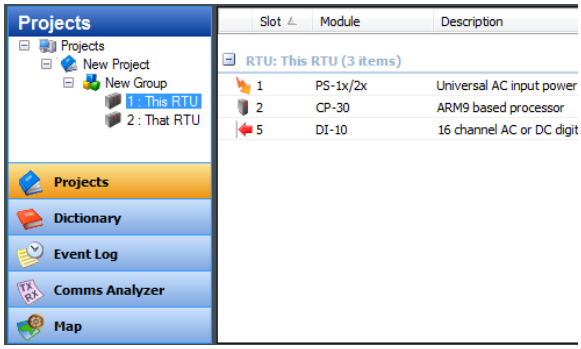
The Recent Projects that were opened in the past can be opened again.

### View RTUs




When a project name is selected in the navigation pane, the RTUs and RTU groups in that project are displayed in the workspace.

### View Modules

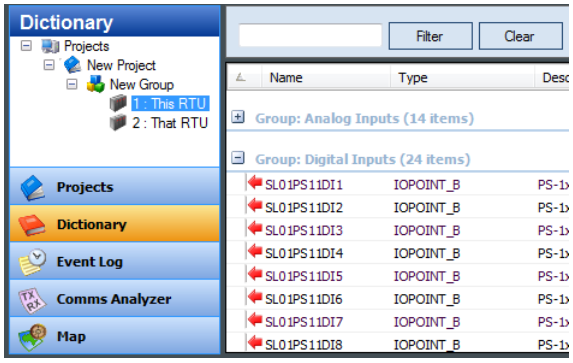



When an RTU name is selected in the navigation pane, the RTU's modules are displayed in the workspace.

This view allows module properties to be configured (double-click on any module).

For more information please see the topic RTU Configuration - Module Properties.

## View Dictionary

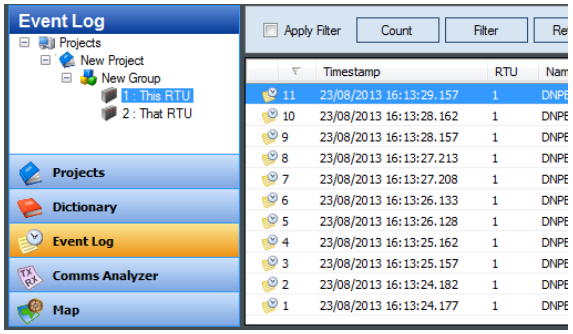



When an RTU name is selected in the navigation pane, the RTU's dictionary is displayed in the workspace.

This view allows variables to be edited (double-click on an existing variable) or created (select the *New* button).

For more information please see the chapter Dictionary.

## Event Log

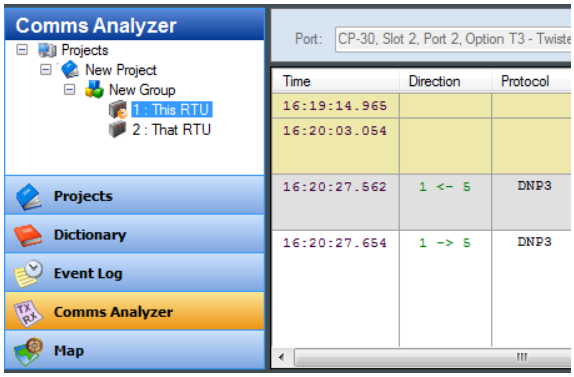



When an RTU name is selected in the navigation pane, the RTU's event log is displayed in the workspace.

This view allows event logs to be retrieved from an RTU, filtered, exported (saved) and cleared.

For more information please see the topic View - Event Logs.

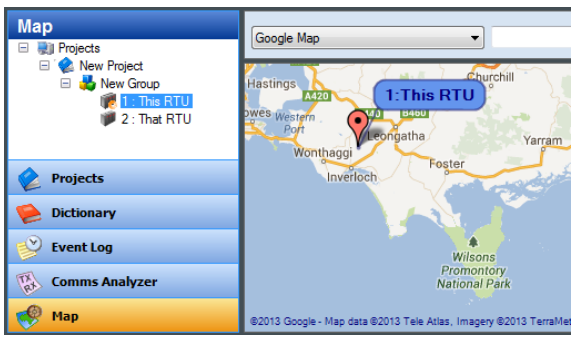
## Comms Analyzer




When an RTU name is selected in the navigation pane, a list of received and transmitted communications messages is displayed in the workspace.

This is useful for diagnosing communications issues. See Comms Analyzer.

## Map




When an RTU name is selected in the navigation pane, the workspace displays the RTU location on a map, if it has been set.

For more information see Maps

# 4. RTU Configuration

## 4.1 Overview

The primary function of Toolbox PLUS+ is to configure one or more Kingfisher RTUs so that they can perform the required functions. This involves specifying:

- the types of power supply, processor, I/O and communications modules
- the communications protocols to use (Modbus, DNP3, etc.)
- routes, which specify the ports to use to communicate with other RTUs or devices
- protocol and I/O points of the required types (boolean, integer, etc.)
- logic processing functions, which are entered using ISaGRAF (ladder logic, structured text, etc.)
- other settings and options (addresses, timeouts, security, etc.)

These settings are saved to a project file on the PC, then compiled into a form which can be downloaded to each RTU's processor module.

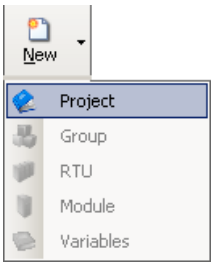
## 4.2 Projects, Groups and RTUs

In Toolbox PLUS+, a system is organised into the following hierarchy:

- A **project** contains all settings for the entire system. This may encompass many RTUs spread across multiple sites.
- The project may optionally define **groups** of RTUs which share something in common, e.g they might be located at the same site.
- An **RTU** represents a physical RTU – either a rack of modules or a standalone unit such as a G30. Each RTU has a unique address. The RTU has various properties that can be set, and various status items that can be viewed.
- An RTU consists of the configured set of **modules** (or internal **cards** in the case of the G30). Like RTUs, individual modules have properties that can be set, and status items that can be viewed.

Projects and groups are only used by Toolbox PLUS+ to organise how the information is displayed and saved. Project and group names are not downloaded into the RTU.

A new project must be created before any RTU configurations can be created.

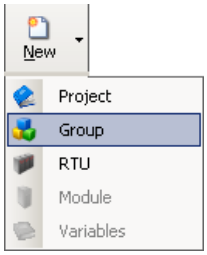



**Create New Project**

Select *New* → *Project*

(Or an existing project can be opened)

When configuring multiple RTUs, they can be kept in groups. Grouping similar RTUs can simplify large project layouts, for example, outstations and master RTUs may be grouped.

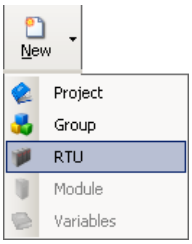



**Create New Group**

Select the project name in the navigation pane

Select *New* → *Group*

RTUs can be added directly into a project or added into a project group.



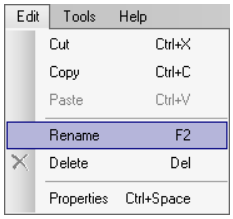

**Create New RTU**

Select the project or group name in the navigation pane

Select *New* → *RTU*

Select the Type of RTU required: CP-30 or G30. A minimal RTU configuration will be created containing power supply and processor modules.

The new project, group and RTUs can be renamed.



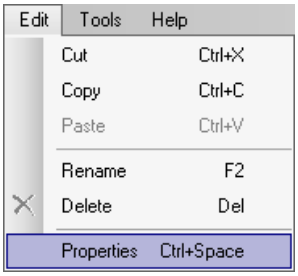

**Rename Project, Group or RTU**

Select the project, group or RTU in the navigation pane.

Select *Edit* → *Rename*

## 4.3 Project Properties

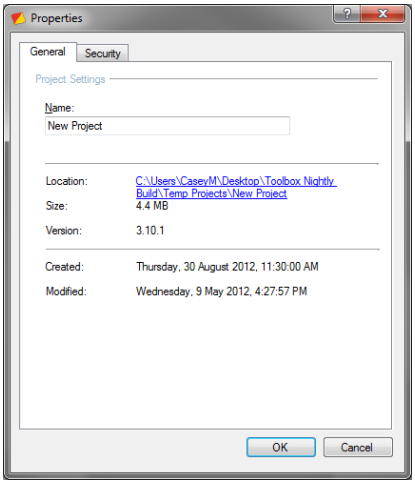
A project has certain properties that can be set. These relate to the overall project file, and do not affect the operation of any RTU.




**Edit Project Properties**

Select the Project name in the navigation pane

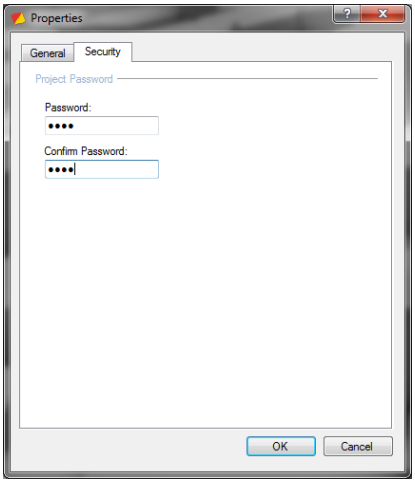
Select *Edit* → *Properties* (or double-click the Project name)




**Name:** (up to 255 characters) a descriptive name for the project can be entered here.

Names can include spaces, hyphens (-), underscores (_), commas (,) and periods (.). Other special characters are not permitted.

If no name is entered then you will be prompted for one when the project is saved to disk.




Project security is a low level of security designed to avoid accidental changes to project files. The password is stored in text format in one of the project databases allowing the project to be recovered in the event of password loss.

**Password, Confirm Password:** (up to 20 characters) When configured, this password will be required in order to open the project in Toolbox PLUS. To disable password protection, clear both parameters.

Note the project must be saved to enable password protection.

## 4.4 Add Modules

### 4.4.1 Backplanes and Racks

Physically, all modules in a CP-30 based RTU are installed on a **backplane**. A backplane has 4, 6 or 12 slots.

An RTU's backplanes are organised into between one and four linked **racks**. Each rack supports up to 16 modules and may contain either one or two linked backplanes. Therefore the maximum number of modules per RTU is 64.

The slot numbers for a given backplane depend on the type of backplane (4, 6 or 12 slot), and the rack number. (The rack number is configured using DIP switches on the backplane.) Rack #1 always contains slots 1-16, Rack #2 contains slots 17-32, Rack #3 contains slots 33-48 and Rack #4 contains slots 49-64.

Each rack consists of either:

- One 6-slot backplane, containing slots 1-6 (or 17-22 for Rack #2, etc.), or
- One 12-slot backplane, containing slots 1-12 (or 17-28 for Rack #2, etc.), or
- One 4-slot backplane, containing slots 13-16 (or 29-32 for Rack #2, etc.), or
- One 12-slot plus one 4-slot backplane, containing slots 1-16 (or 17-32 for Rack #2, etc.)

Note that Toolbox PLUS+ does not distinguish between physical backplanes – it treats the RTU as a single unit with 64 slots, numbered 1-64. It is important, however that each module's slot number be entered correctly. For example, in a small RTU with a single 4-slot backplane, the entered slot numbers should be in the range 13-16, not 1-4.

For more information, refer to the *Kingfisher PLUS+ Hardware Reference Manual*, available for download from the CSE-Semaphore support website.

### 4.4.2 Adding Modules

When you first create an RTU in Toolbox PLUS+, it will contain two modules by default: a power supply module in backplane slot 1 and a processor module in slot 2.

Modules can now be added or moved to build up the desired RTU layout.

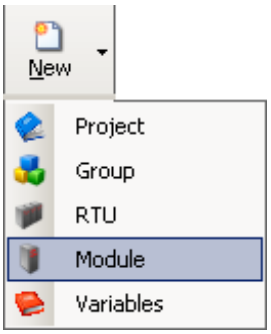
For a CP-30 based RTU, module types that can be added include:

- power supply modules (PS-1x/PS-2x). Multiple power supplies can be included to provide redundancy.
- processor modules (CP-30). At least one CP-30 must be present. A second CP-30 can be included, for redundancy. If two CP-30s are used then one must be in an even slot and the other in an odd slot.
- communications modules (MC-31, or its predecessor, MC-30). These provide additional Ethernet or serial communications ports (up to 3 ports per module)
- I/O modules (AI-1/4, AI-10, AO-2, AO-3, DI-1, DI-10, DI-5, DO-1, DO-2/5/6, IO-2, IO-3, IO 4, IO 5), which provide analog/digital inputs and outputs.

For a G30 standalone RTU, you can specify:

- the type of power supply card fitted (PSO-ACR, PSO-DCU)
- the type of I/O card fitted (IOD-MX2/3/4)

Toolbox PLUS+ will automatically create variables in the dictionary corresponding to all the IO points in each power supply, IO module or card.



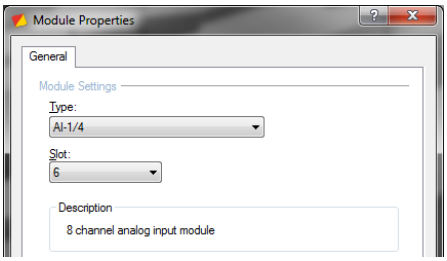

**Add New Module (CP-30) or Card (G30)**

Ensure that *Projects* is selected in the stacked menu bar.

Select the RTU name in the navigation pane

Select *New* → *Module*

(Alternatively, right click on the RTU name and select *New Module*.)

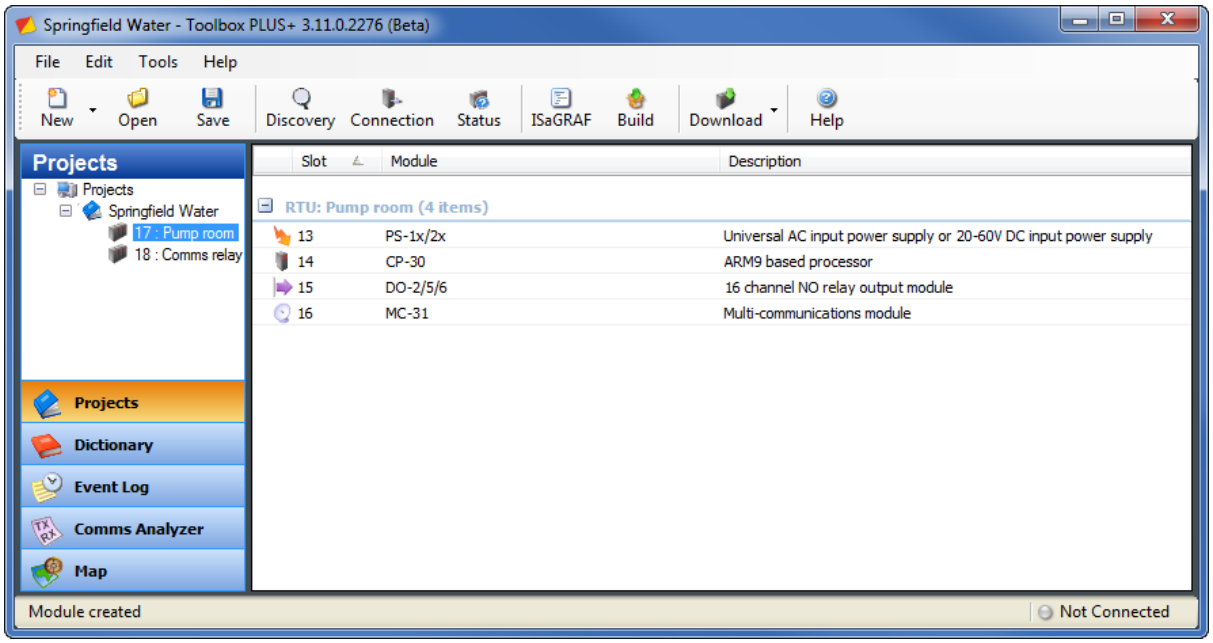



Following the selection, the module type and slot number can be selected, and module properties can be changed.

The slot number is automatically incremented by the software as modules are added and tested for validity.

### 4.4.3 Example

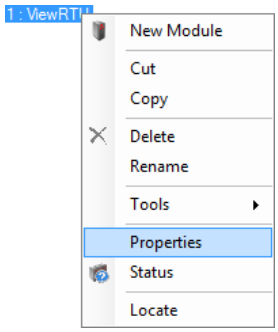
In the following example project, two RTUs have been defined. Details are displayed for RTU #17 (“Pump room”). This RTU consists of a single 4-slot rack (slots 13-16), which contains a power supply, CP-30 processor, digital output module and a communications module.



## 4.5 RTU Properties

The *RTU Properties* menu allows the global settings of each RTU to be modified. The settings are grouped into a number of tabs.

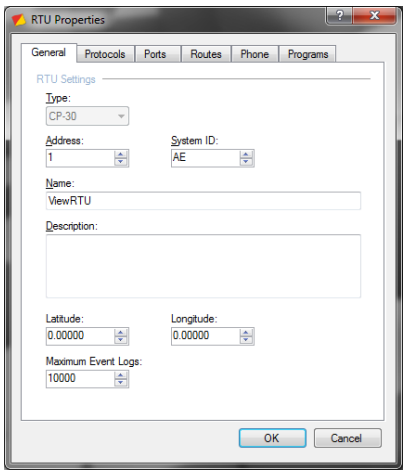
The *General* tab contains descriptive settings (name, location, etc.) and protocol addressing information.




**Edit RTU Properties**

Select the RTU name in the navigation pane

Select Edit → Properties (or double-click the RTU name)




**Address:** (1-65520) Each RTU must have a unique address. If the RTU is connected to a “series 2” network (containing CP12/LP3/G3 based RTUs) then addresses should be set in the range 1-249.

**System ID:** (00 to FF Hex) The communications sync character used to screen incoming Kingfisher protocol messages. An RTU will only respond to messages that have the same sync character as this System ID. It is recommended that the AE default be used except when configuring an RTU to relay radio messages as detailed in the topic RTU Configuration - Routes, Relaying Radio Messages.

**Name:** (0 to 64 characters) Name of the RTU.

**Description:** (0 to 255 characters) Description of the RTU.

**Longitude, Latitude:** (optional) The geographical location of the RTU. Please see the chapter - Maps.

**Maximum Event Logs:** (0-99999, default = 10000) The maximum number of event logs to maintain in Flash memory. Once the maximum limit is reached, the oldest event logs are overwritten.

## 4.6 RTU Properties – Protocols

A protocol is a set of communication commands used to communicate with a device. The RTU supports the following protocols:

| Protocol | Type | Purpose | Port type |
|---|---|---|---|
| Kingfisher | slave, master | RTU configuration and management<br>Data point status and event transfer | Ethernet, serial |
| Modbus/TCP | slave, master | Data point status transfer | Ethernet |
| Modbus/RTU | slave, master | Data point status transfer | Serial |
| Modbus/ASCII | slave, master | Data point status transfer | Serial |
| DNP3 | slave, master | Data point status and event transfer | Ethernet, serial |
| Allen Bradley | master | Data point status transfer | Serial |
| SNMP Client | master | Retrieve RTU status information | Ethernet |
| SNMP Slave | slave | Report RTU status information | Ethernet |
| SNMP Trap | master, slave | Send or receive RTU status notifications | Ethernet |
| User Defined | master, slave | Send or receive raw serial messages | Serial |
| HART | master | Data point status transfer | HART option card |
| NTP | master | Set RTU time from NTP server | Ethernet (CP-30/G30 Port 1 only) |
| VRRP | master, slave | Allow multiple RTUs to emulate a single IP network gateway, for redundancy | Ethernet (CP-30/G30 Port 1 only) |

In the above table, if the RTU implements a “master” protocol then it can initiate messages to another device. These messages are typically triggered when appropriate ISaGRAF custom function blocks are executed. Conversely, when the RTU acts as a “slave” it responds to incoming requests.

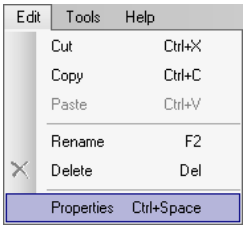
In the above table, a “serial” port type refers to any serial-based option card, including RS232/422/485 (Option I), Fibre (Option F), Dialup (Option D), Line (Option L) and Spread Spectrum Radio (Option R2/R3/R4).

For Ethernet (TCP/IP) based protocols, multiple protocols can generally operate simultaneously on the same physical Ethernet port. Messages are distinguished using the TCP or UDP “port numbers” built into TCP/IP. For example, DNP3 messages are normally directed to TCP port 20000, while Modbus/TCP messages use port 502.

Serial ports can only be configured for a single protocol.

By default, only the Kingfisher protocol is enabled. To set up other protocols, you need to:

- Add the protocol to the RTU configuration. This causes the required driver software to be started when the configuration is downloaded to the RTU.
- Assign the protocol to the required ports. Toolbox PLUS+ validates all selections, and prevents you from assigning two different protocols to a serial port, for example. Note that the HART, NTP and VRRP protocols do not need to be assigned to a port as they operate using fixed ports.




**Add, Remove or Edit Protocol**

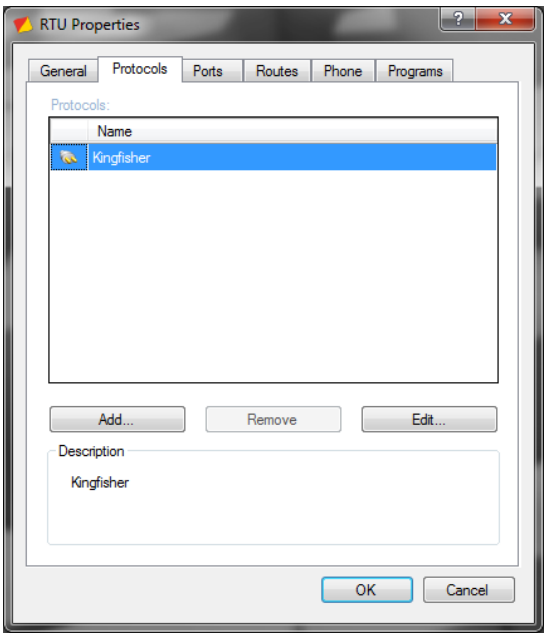
Select the RTU name in the navigation pane

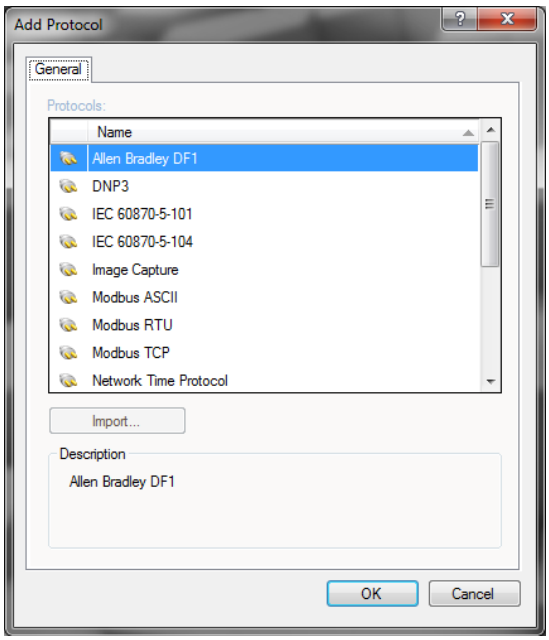
Select *Edit* → *Properties* (or double-click the RTU name)

Select the *Protocols* tab

Select the *Add* button to add a protocol OR select an existing protocol to edit or remove and then select the *Edit* or *Remove* button

Note: multiple protocols can be selected and added at the same time by using the CTRL or SHIFT keys and selecting the relevant protocol(s) with the mouse.




The following sections briefly describe the available protocols

### 4.6.1 Kingfisher

*Overview*

Kingfisher Protocol is the “native” protocol for Kingfisher RTUs. By default, it is enabled on all Ethernet and serial ports.

This protocol allows data to be transmitted through multi-level networks. That is, messages to a remote RTU can be automatically forwarded via intermediate RTU(s).

The protocol supports the transfer of event logs (historical data), as well as real-time data.

Kingfisher Protocol is also used by Toolbox PLUS+ for querying the status of an RTU, e.g. checking firmware version, number of logged events, etc.

The RTU implements both the slave and master ends of the protocol, so an RTU can be set up as a concentrator, which can poll outstation RTUs for events and then be polled by Toolbox PLUS+ or a SCADA system. When operating as a master, the RTU generates Kingfisher messages using custom ISaGRAF function blocks.

### *Kingfisher Variables*

In order to transfer data using the Kingfisher Protocol, registers must be manually created in the Dictionary to hold the data:

- To store local data where they can be polled by another system, Local Kingfisher Registers need to be created. These have names of the form KFR*n.*
- To store data that have originated from another RTU, Network Kingfisher Registers should be created. These have names of the form KF*r*R*n*, where *r* is the RTU from which they originated.

The Kingfisher Protocol supports two types of polling (In this example RTU1 is polling RTU2):

- Direct polling: The KF_RX_DATA function block will copy RTU2's local registers KFR*n* into network registers KR2R*n* on RTU1.
- Indirect polling: The KF_NW_RX_DATA function block will copy RTU2's network registers KF3R*n* and KFR4*n* (assuming RTU2 is set up to poll RTU3 and RTU4) into RTU1's matching network registers KF3R*n* and KFR4*n*.

### *Port Types*

The Kingfisher protocol is supported on all port types, except the HART option card.

For Ethernet ports, Kingfisher protocol uses UDP ports 473 and 4058.

## 4.6.2 Modbus

### *Overview*

Modbus is a simple, widely used protocol which can transfer integer and boolean values. Modbus does not support the transfer of historical event data. The Kingfisher RTU supports three Modbus variants:

- Modbus/RTU is used on serial lines, e.g. RS232, RS485
- Modbus/ASCII is also used on serial lines but it uses ASCII encoding, which is less efficient but can be easier to deal with in some systems
- Modbus/TCP uses Ethernet to transport the Modbus packets over a TCP/IP network. TCP port 502 is used.

For each of these, the RTU supports both the master end (where the RTU initiates a request when the MODBUS custom ISaGRAF function block is executed), and the slave end (where the RTU responds to incoming requests).

### *Data Types*

Modbus defines four types of data point:

- **Coils** are single bit outputs
- **Discretes** are single bit inputs
- **Holding registers** are 16-bit output registers
- **Input registers** are 16-bit input registers

A Modbus device may have up to 65536 of each type of point. These are addressed using a 16-bit index (0-65535).

*Modbus Variables*

In order to transfer data using Modbus, variables must be created in the Dictionary to hold the data:

- To store local data where they can be polled by another system, Local Modbus Registers need to be created. These have names of the form MODC*n* (coil), MODD*n* (discrete), MODH*n* (holding) or MODI*n* (input).
- To store data that have originated from another RTU, Network Modbus Registers should be created. These have names of the form MOD*r*C*n* (coil), MOD*r*D*n* (discrete), MOD*r*H*n* (holding) or MOD*r*I*n* (input), where *r* is the RTU from which they originated.

Note that Modbus addresses are 8 bits long (1-254). When accessing Modbus variables on an RTU with an address greater than 255, be aware that only the least significant byte (lower 8 bits) of the RTU address will be used in Modbus messages.

For example, if you have slave RTUs with address 20 (0014h) and address 276 (0114h) on the same multi-drop network (e.g. Ethernet or RS-485), then you will not be able to poll both of them using Modbus, because the least significant byte of each address is the same. To rectify this you would need to change one of the addresses, or use different physical ports.

### 4.6.3 DNP3

*Overview*

Distributed Network Protocol version 3 (DNP3) is a widely used telemetry protocol. It is more sophisticated than Modbus in that it supports:

- polling for events (state changes) as well as current values (Class 0 data)
- optional unsolicited reporting of state changes from slave to master, which reduces the amount of polling required
- grouping events into classes (Class 1, 2 or 3) which can be selectively retrieved
- a richer set of data object types

A Kingfisher RTU can act as a DNP3 Slave, a DNP3 Master or both. The RTU can respond to DNP3 messages (DNP3 Slave), initiate DNP3 messages using DNP3 function blocks (DNP3 Master) or forward DNP3 messages. DNP3 has various protocol settings that can be edited as detailed below.

*Data Types*

The following DNP3 data types are supported:

- **Analog inputs** are 32-bit integer (variations 1 and 3), 16-bit integer (variations 2 and 4) or 32-bit floating point (variation 5) input registers
- **Analog outputs** are 32-bit integer (variation 1) or 16-bit integer (variation 2) or 32-bit floating point (variation 3) output registers
- **Binary inputs** are single bit inputs (variations 1 and 2)

- **Binary outputs** are control outputs (variations 1 and 2). Single bit pulse on/off and latch on/off operations are supported, as well as paired trip/close operation, where one physical output “trips” the device (turns it off) and the other “closes” it (turns it on)
- **Binary counters** are 32-bit integer (variations 1 and 5), 16-bit integer (variations 2 and 6) counter inputs
- **Frozen counters** are copied from the associated binary counter when a DNP3 “freeze” command is received

### *DNP Variables*

In order to transfer data using DNP3, variables must be created in the Dictionary to hold the data:

- To store local data where they can be polled by another system, Local DNP3 Registers need to be created. These have names of the form DNPAI*n* (analog input), DNPAO*n* (analog output), DNPBI*n* (binary input), DNPBO*n* (binary output), DNPBC*n* (binary counter) or DNPFC*n* (frozen counter).
- To store data that have originated from another RTU, Network DNP3 Registers should be created. These have names of the form DNP*r*AI*n* (analog input), DNP*r*AO*n* (analog output), DNP*r*BI*n* (binary input), DNP*r*BO*n* (binary output), DNP*r*BC*n* (binary counter) or DNP*r*BC*n* (frozen counter), where *r* is the RTU from which they originated.

Note that for DNP3, another way to create a specified number of variables is by adjusting the settings on the DNP3 Protocol settings *General* tab (such as *Number of Binary Inputs*).

If the RTU only needs to forward DNP3 messages, DNP3 variables do not need to be configured. DNP3 messages will be forwarded if a route has been configured for the target RTU and the DNP3 protocol is enabled on that port. The communication timeout and retry parameters associated with this route are applied to the DNP3 messages forwarded through the RTU.

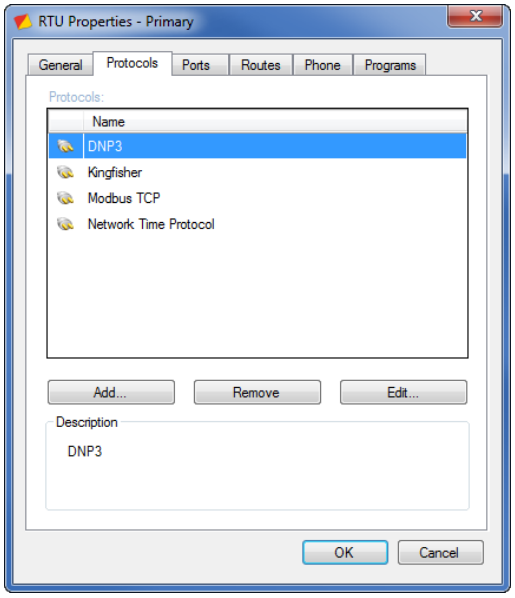
### *Port Types*

DNP3 is supported on all port types, except the HART option card.

For Ethernet ports, DNP normally uses TCP port 20000. It can also be configured to use UDP port 20000

### *Settings*

DNP3 has a number of additional settings that can be adjusted:



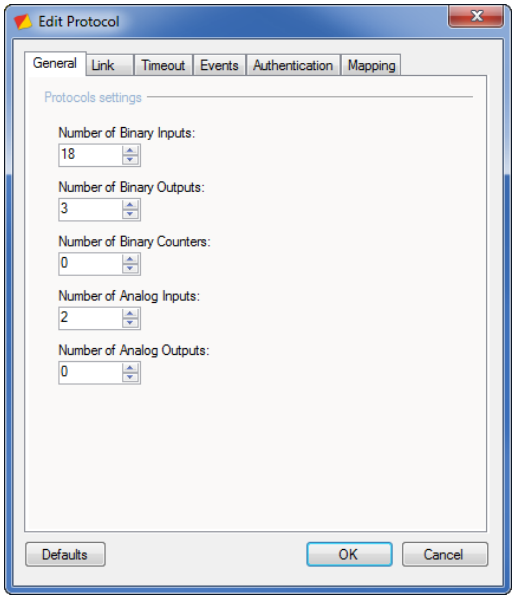

**DNP3 Settings**

Select the RTU name in the navigation pane

Select *Edit* → *Properties* (or double-click the RTU name)

Select the *Protocols* tab

Select *DNP3* from the protocols list and click *Edit* (or just double-click on *DNP3*)

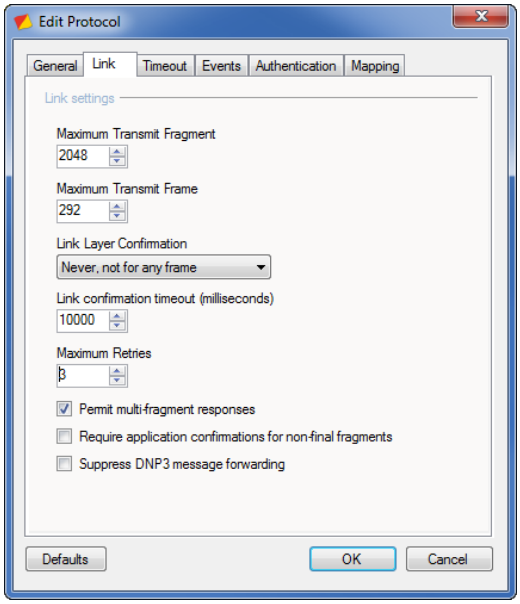



DNP3 variables for the local RTU can be automatically created in the Dictionary (or deleted) according to the settings on the *General* tab.

If you change one of these values and press OK, then a consecutive sequence of variables of the specified type will be created. This may cause existing variables to be deleted or renumbered.

If binary counters are defined, you can also create frozen counter variables (DNPFC*n)* for some or all of them, using the Dictionary. (Binary Counter values are copied into the corresponding Frozen Counters following the appropriate DNP3 “freeze counters” command.)

**Defaults:** This button allows you to specify the settings to use (e.g. class and variation) for any new DNP3 variables that are created.




**Maximum Transmit Fragment:** (14-2048, default=2048) This only needs to be changed if the destination device cannot handle large transmit fragments (as detailed in the DNP3 device profile documents).

**Maximum Transmit Frame:** (14-292, default=292) Multiple frames are used to transmit one fragment. This setting only needs to be changed if the communications device has a message size or frame limitation.

**Link Layer Confirmation:** (default=Never) If this setting is enabled then each transmitted DNP3 frame will be marked as needing to be acknowledged by the receiver; if an acknowledgment is not received within the timeout period then the frame will be automatically retried.

Link Layer Confirmation is useful where DNP3 is used on an “unreliable” link, i.e. one where there is no underlying error detection protocol. This includes serial connections, and UDP connections over Ethernet. Link Layer Confirmation is not required for TCP based links (which are the default when DNP3 is used over Ethernet).

**Link confirmation timeout (milliseconds):** (1-65535, default=10000ms) Only used if Link Layer Confirmation is enabled.

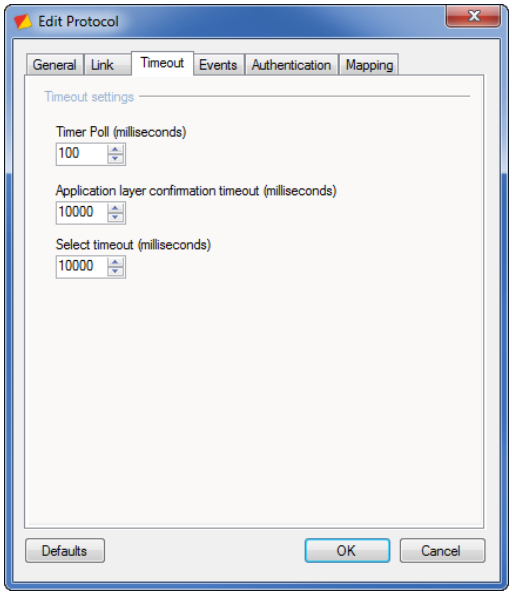
Note: If you are using the KF_GET_PENDING function block to control the transmission of poll messages, and link layer confirmations are enabled, then this timeout should be set to a value which is less than or equal to the Route timeout (see RTU Properties – Routes). If this is not done then following an unanswered poll the Pending flag will be reset after the Route timeout expires (enabling the next poll to proceed in logic), but the request will not actually be sent until the Link confirmation timeout expires.

**Maximum Retries:** (0-65000, default=3) Number of retries if a link layer confirmation timeout occurs.

**Permit multi-fragment responses:** (default=enabled).

**Require application confirmations for non-final fragments:** (default=disabled)

**Suppress DNP3 Forwarding:** (default=disabled, i.e. message forwarding is enabled) If indirect routes are used, Kingfisher RTUs will automatically forward DNP3 messages if the target RTU exists in the route table. This can be problematic when network latency is introduced into a system and can cause errors.

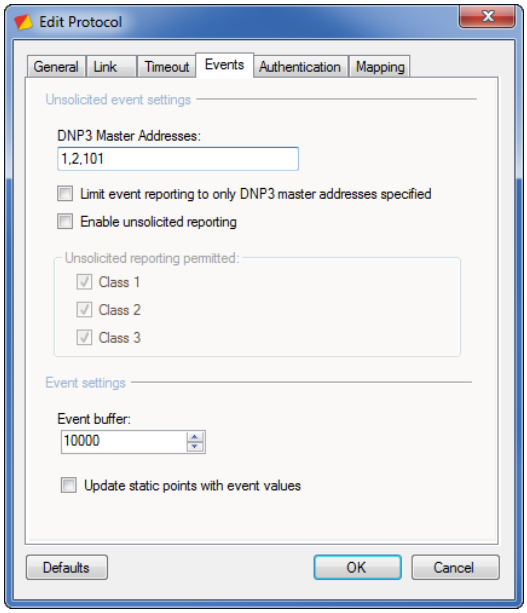



The following settings apply when the RTU is operating as a DNP3 slave.

**Timer Poll (milliseconds):** (1-65535) Specifies how often to check the static value of every DNP3 variable for change. Events are created for any variable that has changed and has a class assignment of 1, 2 or 3.

**Application layer confirmation timeout (milliseconds):** (1-65535, default=10000ms) Specifies how long to wait for a confirmation that event data has been received by the master.

**Select timeout (milliseconds):** (1-65535, default=10000) Specifies how long to wait for an *Operate* command after receiving a *Select* command.




The following settings apply when the RTU is operating as a DNP3 slave.

**DNP3 Master Addresses:** (0-65535) A comma separated list of DNP3 master addresses (see below).

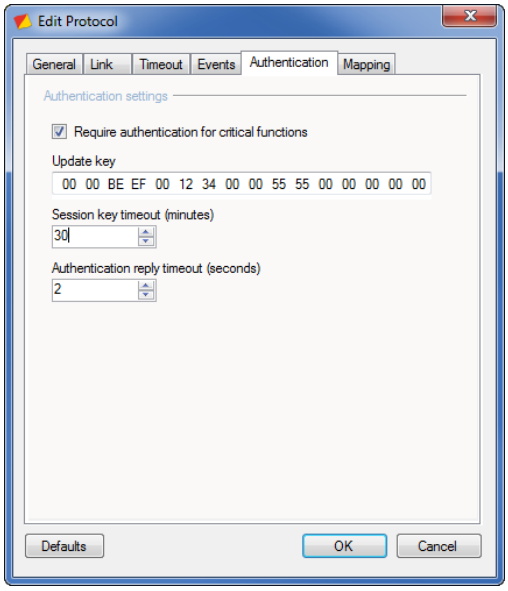
**Limit event reporting to only DNP3 master addresses specified:** (default=disabled) If enabled, events will only be reported to the specified list of DNP3 masters. Status responses and static variable values will still be reported to any master.

**Enable unsolicited reporting:** (default=disabled) When enabled, the RTU will automatically send an unsolicited report if there are new Class 1, 2 or 3 events (when enabled respectively). These will be sent to the specified list of master addresses.

Note that this setting specifies the initial state of unsolicited reporting. During operation, it may be overridden by a request from the DNP3 master, or by executing the DNPS_UNSOL_ENABLE or DNPS_UNSOL_DISABLE function blocks.

**Event buffer:** (default=10000) The maximum number of un-reported events of each data type to keep in the DNP3 event buffer. If this limit is exceeded, the oldest events will be overwritten and the buffer overflow flag will be set in the Internal Indication Register.

**Update static points with event data:** (default=disabled) If enabled, the RTU will update its static DNP3 variables based on event data.



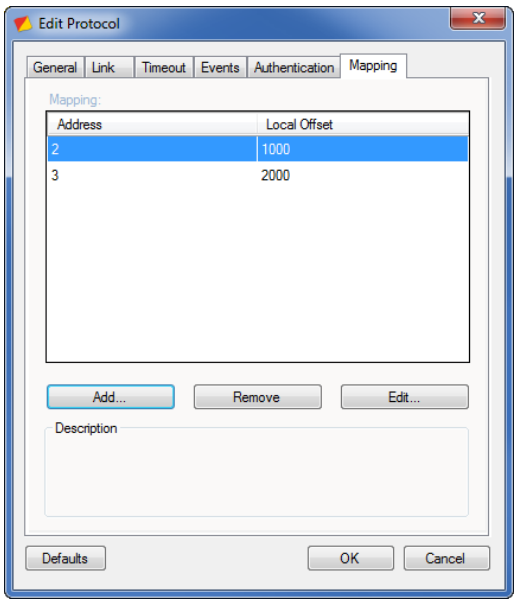

**Require authentication for critical functions:** (default=disabled) Used for DNP3 Slave RTUs when security is required. When enabled, an Update key can be entered (consisting of 16 hexadecimal bytes). This Update key must then be provided by a DNP3 master device before it can request a critical function.

For a DNP Master RTU, authentication is configured for each route that is used to communicate with a Secure DNP3 Slave RTU.

Critical functions requiring authentication (according to the DNP3 Secure Authentication standard) are: Write; Select; Operate; Direct Operate; Direct Operate No Acknowledgement; Cold restart; Warm Restart; Initialise Application; Start Application; Stop Application; Enable Unsolicited Messages; Disable Unsolicited Messages; Record Current Time; Authenticate; and Activate Configuration.

**Session key timeout:** (0-99999 minutes) The Update key is used to create an initial session key. The session key is automatically changed each Session Key Timeout interval to protect against “replay” attacks.

**Authentication reply timeout:** (2-600 seconds) How long to wait for a reply to the initial authentication message.




**Mapping** is useful for data concentration. DNP3 objects obtained from remote RTUs can be stored as if they are objects in the local RTU.

To prevent local and remote objects clashing, objects from the remote RTU are given an offset.

Select the *Add* button to add a new mapping or select an existing mapping and then select *Edit* or *Remove.*

Example: The local RTU has address 1. A Mapping is configured with Address=2 and Local Offset=1000. When DNP objects are polled from remote RTU2, the objects are stored in RTU1 as follows:

Remote RTU2 to Local RTU1

DNP2BI1 → DNPBI1001
DNP2AI1 → DNPAI1001
etc...

### 4.6.4 Allen Bradley DF-1

This protocol allows serial communications with an SLC500 PLC using the DF-1 protocol in half duplex slave mode. The data format is 8 data bits, 1 stop bit and no parity. The RTU always operates as the master – messages are generated using custom ISaGRAF function blocks.

Data read using the Allen Bradley protocol are transferred using a block of local Kingfisher registers (KFR*nn*).

### 4.6.5 SNMP

Simple Network Management Protocol or SNMP is supported on the processor Ethernet ports. The RTU supports the following modes of operation:

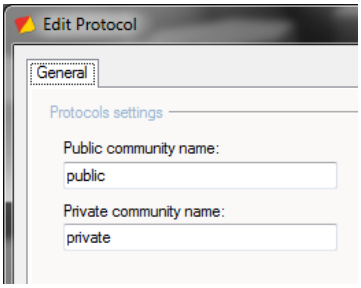
- The RTU can be an SMTP Manager (master), and can send query messages to other devices using custom ISaGRAF function blocks.
- The RTU can be an SMTP Agent (slave), where it responds to incoming queries.
- The RTU can send or receive SMTP Trap messages (asynchronous notifications) using ISaGRAF custom function blocks.

Note that each of the above is treated as a separate protocol, and must be added and enabled separately.

When operating as a slave, the RTU makes available certain configuration and status information. These are defined in the Semaphore MIB (Management Information Block) document, which is available on the CSE-Semaphore support website.The SNMP object identifier (OID) codes for Kingfisher RTUs are in the range 1.3.6.1.4.1.27982.1.*n.n.n*. These data include:

- RTU address and name
- Event log information
- Network interface and traffic information

The SNMP Slave Daemon has the following protocol settings that can be edited. To bring up this dialog, double click on *SNMP Daemon (Slave)* in the RTU Properties protocol list.




**Public community name:** (default=”public”)

**Private community name:** (default=”private”)

These are effectively passwords that can be used to restrict access to the information that the RTU makes available via SNMP. Data will be made available if the client specifies either of the configured passwords.

### 4.6.6 HART

The HART protocol can only be used with a HART option board and is automatically added to the RTU when a HART port is added to the RTU configuration.

The RTU supports the master end of this protocol. Messages can be initiated by the RTU using the HART function block, which is based on Revision 5 of the Hart protocol.

HART data is stored in a block of integer and floating point Kingfisher network variables (KF*r*R*n* and KF*r*F*n*), where *r* is user-specified and not necessarily the same as the device's actual

### 4.6.7 User Defined

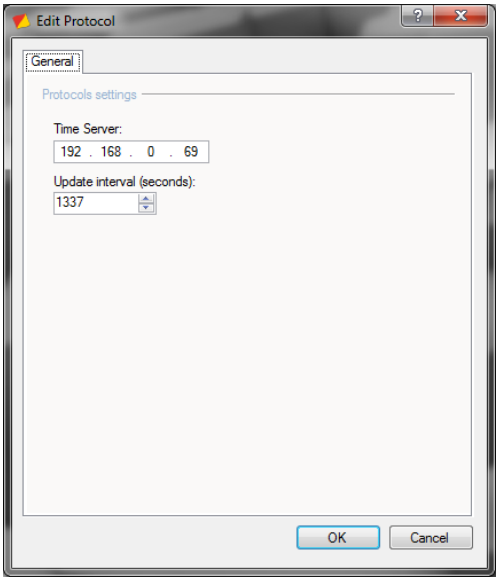
The User Defined ISaGRAF function blocks can be used to send and receive arbitrary serial messages. This allows simple serial protocols to be implemented in the ISaGRAF logic program.

### 4.6.8 NTP

The Network Time Protocol (NTP) allows the RTU time to be periodically synchronized with a local or remote time server.

This protocol is supported on the CP-30/G30 main Ethernet port, and on CP30 ports fitted with a T3 option card. It does not need to be specifically enabled on the port.

The following parameters will need to be set. To bring up this dialog, double click on *Network Time Protocol* in the RTU Properties protocol list.




**Time Server:** IP address of the local or remote time server.

**Update Interval (seconds):** How often to check and update the RTU time.

### 4.6.9 VRRP

A large telemetry system which uses TCP/IP networking may be set up so that a number of slave RTUs on a LAN all use one master RTU as the gateway to the wider network. That is, on each slave RTU the Ethernet port Gateway IP address is set to that of the master RTU.

If the gateway RTU (or its network interface) fails, all of the slave RTUs will no longer be able to communicate outside the LAN.

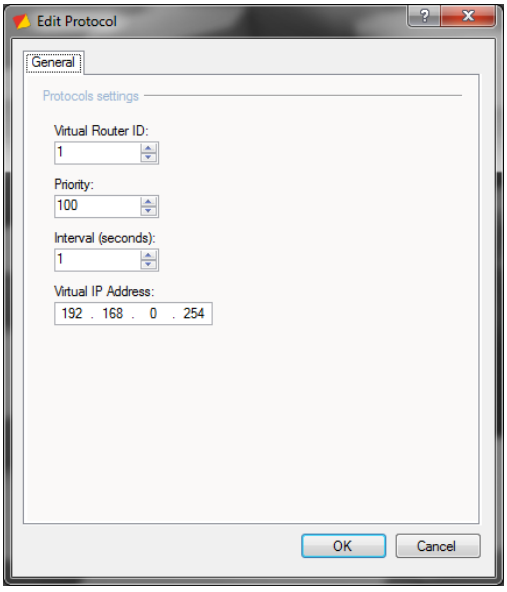
The Virtual Router Redundancy Protocol (VRRP) is designed to allow multiple RTUs to act as a set of redundant gateways, which then appear to be a single “virtual router”.

At any one time, one of the RTUs with VRRP enabled will be the active gateway. It will use a configured IP address (which the slave RTUs will be configured to use as their gateway address) and a special hardware (MAC) address (00:00:5E:00:01:*xx* where *xx* = the configured “Virtual Router ID”) on its main Ethernet port.

The active gateway RTU will also send out periodic broadcasts (to the multicast address 224.0.0.18), which indicate to the standby gateway RTU(s) that it is still functional. If these broadcasts are no longer detected then a standby gateway RTU will configure its Ethernet port to use the configured gateway IP address and the special MAC address and will therefore seamlessly take over the routing function.

The active/standby VRRP status of a gateway RTU can be checked using the VRRP function block in an ISaGRAF logic program.

To use this feature, two or more gateway RTUs need to have the VRRP protocol enabled, and then configured using the following settings:




**Virtual Router ID:** (1-255) This is an arbitrary identifier. It must be set to the same value on all gateway RTUs.

**Priority:** (1-255) This is used if there are multiple standby gateway RTUs, to decide which one will become active.

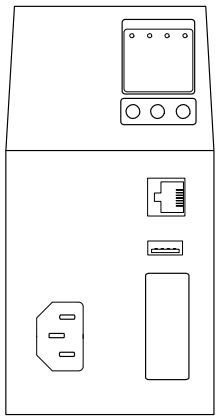
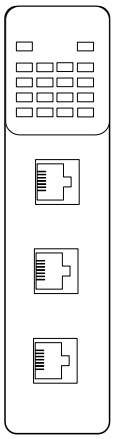
**Interval (seconds):** The interval used by the active gateway RTU when sending the periodic VRRP broadcasts.

**Virtual IP Address:** The IP Address to be shared between the gateway RTUs. All slave RTUs should be configured to use this address as their gateway address.

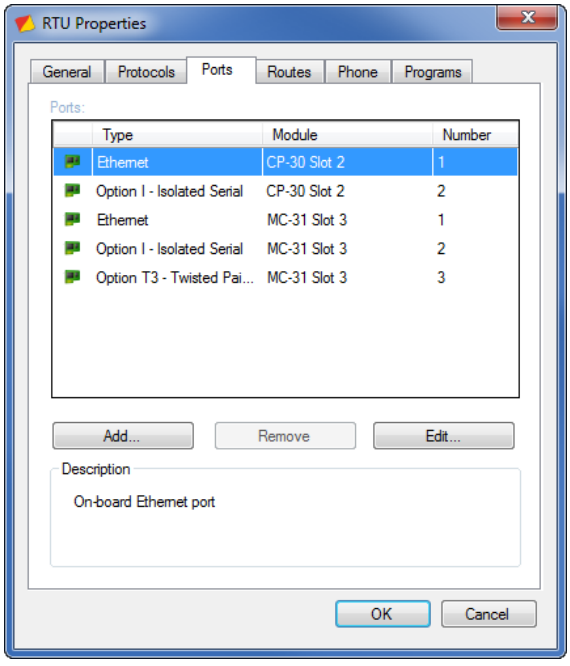
## 4.7 RTU Properties – Ports

The *Ports* tab on the *RTU Properties* dialog is used to configure the physical communications ports in the RTU. These include Ethernet and serial ports, and include ports on the processor module (CP-30/G30) and on communications modules (MC-31).

A G30 based RTU can support 2 communications ports (one Ethernet, one serial), while a CP-30 based RTU can address up to 192 ports (although performance is not guaranteed and users need to consider bandwidth limitations and practice common sense with high port counts).

| G30 RTU | CP-30 and MC-31 Module |
|---|---|
|  Port 1 – Ethernet<br>Port 2 – USB Storage<br>Port 3 – Option Board (Serial) |  Port 1 – Ethernet<br>Port 2 – Option Board (Serial or Ethernet)<br>Port 3 – Option Board (Serial or Ethernet) |

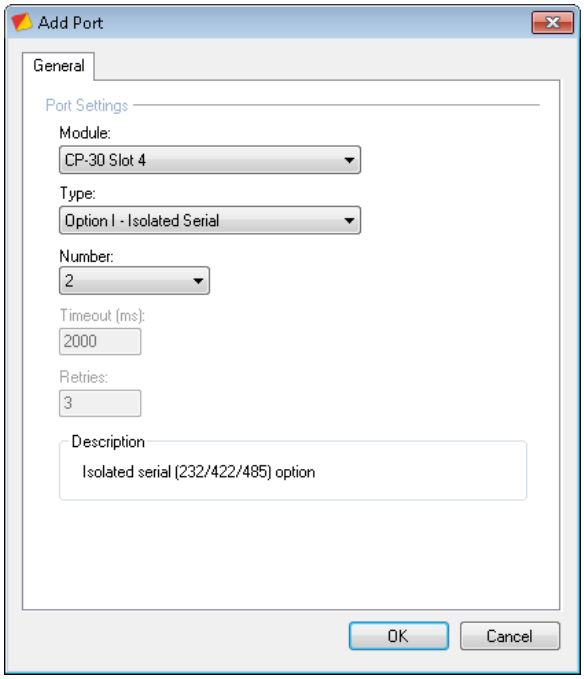
The *Ports* tab will initially show a list of all fixed ports, i.e. the Ethernet port (Port 1) on the CP-30 and each defined MC-31 module. If option card ports are present, you can add these by clicking the *Add* button.




### Add a Port

Select the *Add* button to add an option card port.

If you want to change the type or location of an existing option card port then click *Remove*, then re-add it.




**Select Port Type**

Define the type and location of the option card.

**Module:** Select the CP-30 or MC-31 module containing the option card.

**Type:** Select the type of option card. The various different types of port are described below.

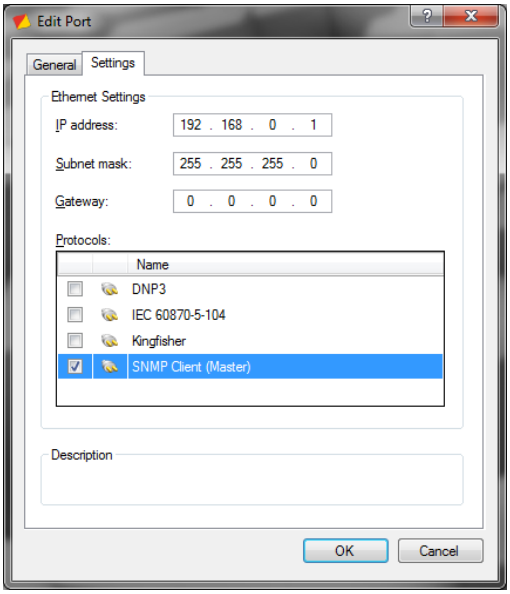
**Number:** Specify the port number within the CP-30 or MC-31 module (2 or 3)

Once the required ports have been added, most will require various settings to be configured. These are described in the following sections.

### 4.7.1 Ethernet

A twisted pair 10/100BaseT Ethernet port is included as Port 1 on the CP-30, G30 and MC-31. Additional Ethernet ports may be added to the RTU by installing T3 (twisted pair) or A3 (fibre optic) option cards in Ports 2 or 3 of a CP-30 or MC-31.

To configure Ethernet, double click on the required Ethernet port in the port list on the *Ports* tab on the *RTU Properties* dialog (or click *Edit*). The following settings are available:




**IP Address:** (default=192.168.0.1) Each Ethernet port must be assigned a valid IP (internet protocol) address which is suitable for the network to which it is connected (check with your network administrator). The assigned IP address must be unique on the local network. In most cases the RTU will be connected to a private LAN, in which case the IP address will be allocated from one of the private address ranges: 10.x.x.x, 172.16-31.x.x or 192.168.x.x

Note: If there are multiple Ethernet ports on a CP-30 or MC-31 module, then each port must be set to a different subnet, which usually means they must be connected to physically separate networks. For example, if Port 1 uses the default IP address and subnet mask (192.168.0.1/255.255.255.0) then Port 2 or 3's IP address cannot be set to 192.168.0.x (192.168.1.x would be OK).

It is also possible to connect a PC directly to an RTU Ethernet port, using a cross-over cable. In this case, the

RTU can be set to any address, provided that the PC's IP address is set to a different address on the same subnet (e.g. if the RTU Ethernet port is left on the default setting of 192.168.0.1 then the PC could to be set to, say, 192.168.0.2).

**Subnet Mask:** (Default=255.255.255.0) This should be set to the correct subnet mask for the network to which the RTU is connected (check with your network administrator). The subnet mask defines which part of the IP address is used to identify the network (or "subnet") and which part is used to identify a particular device on the network. With the default setting, the first three numbers in the IP address specify the subnet and the last one specifies one of 256 devices connected to that subnet.

**Gateway:** (Default=0.0.0.0, i.e no gateway) This specifies the IP address of a device on the local subnet which is able to forward messages to other networks, or the Internet. If not set then the RTU will only be able to communicate devices that are connected to the local subnet.

A gateway can only be set for Port 1. If multiple gateways are required then an MC-31 would be required, and the second gateway would be configured on port 1 of that module.

For example, multiple gateways would be required in applications where the RTU had, for redundancy purposes, two diverse methods for connecting to the outside world. For example, the primary link could be an ADSL modem connected to Port 1 on the CP-30, which would be configured with the ADSL modem's IP address as the gateway address. The backup link might be a 3G modem connected to Port 1 of the MC-31, which would be configured with its gateway set to the 3G modem's IP address.

**Protocols:** Protocols which have already been added to the RTU are assigned to the Ethernet port by ticking the appropriate box(es). Note that MC-31 Ethernet ports only support Kingfisher, DNP3 and Modbus protocols. CP-30 Ethernet ports support any protocols.

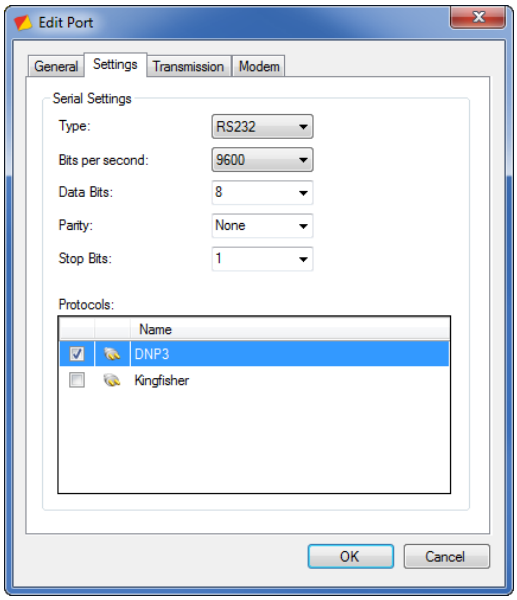
Tip: The Ethernet Settings for the main processor module can be viewed by pressing **F12**.

### 4.7.2 Serial (RS232/422/485)

Serial ports may be added to the RTU by installing Option I (isolated serial) cards in Port 2 or 3 on CP-30 or MC-31 modules. For G30, an OPT-SER card may be installed in Port 3. The following electrical standards are supported:

- RS232 uses single-ended signalling and provides point-to-point communication over relatively short distances (typically up to 15m, depending on baud rate). The link is full-duplex (separate wires for each data direction). RTS and CTS signals are available on the connector and can be used for flow control.
- RS422 uses differential signalling to provide point-to-point or multi-drop communication over longer distances (typically up to 1000m, depending on baud rate). The link is full-duplex (separate pairs for each data direction). The RTS signal is used to enable the transmitter.
- RS485 also uses differential signalling, but is half-duplex. A single pair of wires is used to connect multiple devices in a multi-drop configuration. Protocols used over RS485 must ensure that only one device transmits at a time. The RTS signal is used to enable the transmitter.

To change the serial port settings, double click on the port in the port list on the *Ports* tab.




**Type:** (default=RS232) Specifies the electrical standard (RS232/422/485).

**Bits per second:** The speed at which the RTU will send or receive messages (75-115200bps).

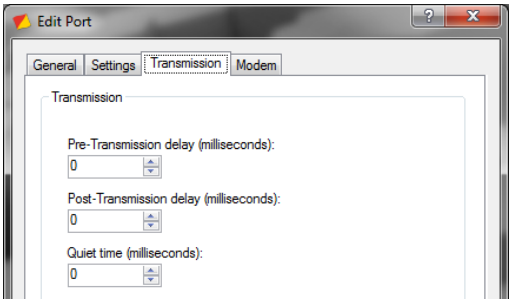
**Data Bits:** Number of data bits transmitted per byte. Must be 8 for any protocol that transmits 8-bit data.

**Parity:** Specifies whether even, odd or no parity is used.

**Stop Bits:** Specifies whether 1 or 2 stop bits are used.

Ensure that the above settings match those of the device with which you are communicating. Note also that for Modbus/ASCII protocol, these settings will be ignored and the following fixed settings used: 7 data bits, Even parity, 1 stop bit.

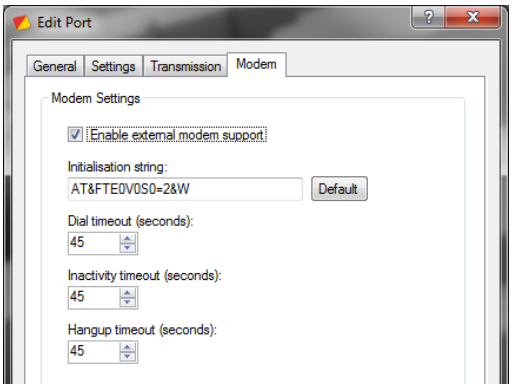
**Protocols:** A protocol already added to the RTU can be enabled on the port. Only one protocol can be enabled for serial ports.




**Pre-Transmission delay (milliseconds):** (default=0) Specifies how long the RTS signal is asserted for before data is sent. Typically set to zero for RS232 or 10 ms for RS485 and RS422.

**Post-Transmission delay (milliseconds):** (default=0) Specifies how long RTS is asserted for after the data has been sent

**Quiet time (milliseconds):** (default=0) The RTU will wait until the line has been quiet (no received messages) for the specified time before sending a message.




**Enable external modem support:** (default=disabled) When enabled, allows an external PSTN modem to be connected to the serial port. Modem settings can then be configured as for a Dialup port as detailed below.

### 4.7.3 Serial (Fibre Optic)

The Option F card is the same as the serial (Option I) card, except that a fibre optic physical interface is used. This allows transmission over distances up to 4km.

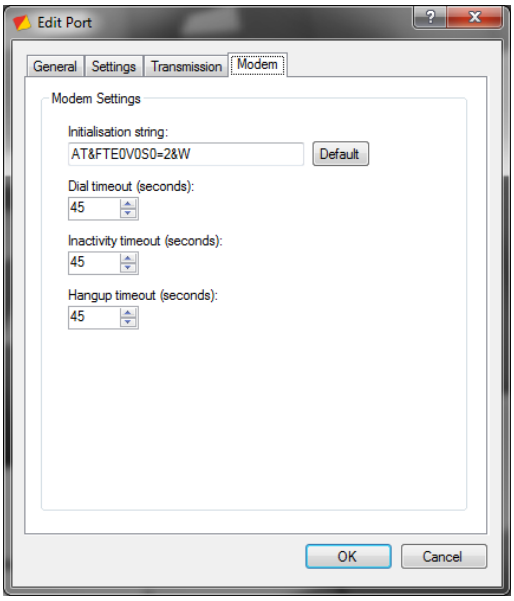
### 4.7.4 Dialup

The Dialup option board (Option D) incorporates a 33.6 kbps PSTN modem.

The basic serial port settings on the *Settings* tab are the same as for the Serial option cards – you need to specify the baud rate and other serial parameters, and select the protocol (only one) that is assigned to the port.

For a PSTN modem, the settings on the *Transmission* tab are not relevant and should normally all be left on zero.

The *Modem* tab contains settings relating to the modem. These are also used when connecting to an external modem using a serial option card.




**Initialisation string:** (default: AT&FTE0V0S0=2&W) These characters are sent to the modem to initialise it after start-up, after disconnection, and if a dial attempt fails. The default string will load factory default settings (&F), select tone dialling (T), switch off command echo (E0), select numeric response codes (V0), enable auto-answer (S0=2) and then save the settings (&W)

Note: X3 (disable dial tone detection) can be added to the default string if the modem is unable to recognize the dial tone or is experiencing problems establishing a connection i.e. AT&FTE0V0S0=2X3&W

**Dial timeout (seconds):** (0-10000) The time to wait from the time when dialling begins until a Carrier Detect indication is received. When dialling a GSM modem, the Dial Timeout should be set to at least 45 seconds.

**Inactivity timeout (seconds):** (0-10000) The RTU will hang up after this amount of time has elapsed since the last message was received. A value of 0 disables the function.

**Hang-up timeout (seconds):** (0-10000) The RTU will hang up after this amount of time has elapsed after connection or after sending the last message. A value of 0 disables this function.

#### *Remaining Online*

To remain online after connection, set Inactivity timeout to 0 and Hang-up timeout to 0 in *both* RTUs in the dialup link. If the line is disconnected, the RTU will reconnect when the next TX or RX message is initiated from ladder logic.

#### *Dialling a Paging Service*

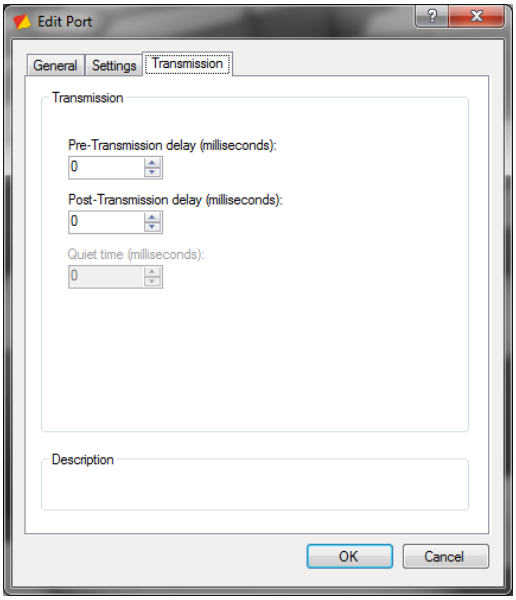
If experiencing problems, error correction may need to be disabled by including '\N0' i.e. AT&FE0V0S0=2\N0&W. If experiencing problems when using an MC module and a Dial option board, the baud rate may need to be limited to 9600 by including F8 in the initialisation string i.e. AT&FE0V0S0=2F8&W.

### 4.7.5 Line

The Line option card (Option L) is used to connect to a private line or analog radio. The board is optically isolated, operates at 1200 bps and provides FSK CCITT V.23 modulation.

The basic serial parameters (*Settings* tab) are fixed at 1200 baud for the Line option card. The desired protocol will still need to be selected.

The transmit timing parameters (*Transmission* tab) may need to be adjusted when using the Line option card:




**Pre-Transmission delay (milliseconds):** How long the carrier and RTS are transmitted for before data is sent. Analog radios typically require a Pre-Transmission delay of 300 ms while private lines use 50 to 100 ms.

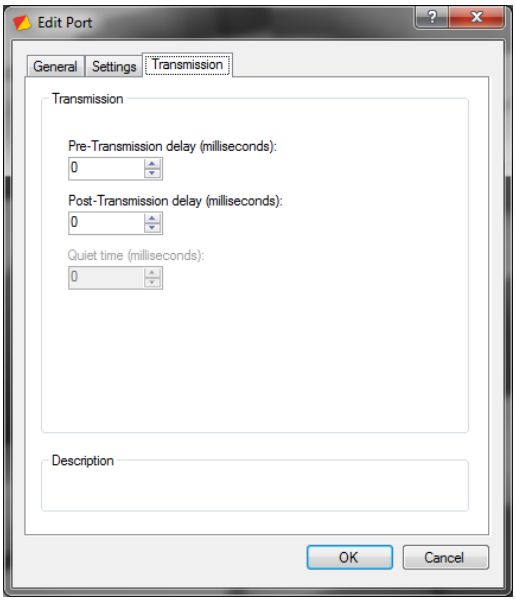
**Post-Transmission delay (milliseconds):** How long the carrier and RTS are transmitted for after the data has been sent. Analog radios typically require a Post-Transmission delay of 100 ms while 50 ms is used for private lines.

### 4.7.6 HART

The HART option board provides a Bell 202 interface to devices supporting the HART protocol. Each HART option board can communicate with up to 15 HART devices.

The basic serial parameters (*Settings* tab) are fixed at 1200 baud for the HART option card. The HART protocol is the only one supported on this port type, so it will be selected automatically.

The transmit timing parameters (*Transmission* tab) may need to be adjusted when using the HART option card:




**Pre-Transmission delay (milliseconds):** How long the carrier and RTS are transmitted for before data is sent. Set to 10 ms or greater to suit the particular HART device.

**Post-Transmission delay (milliseconds):** How long the carrier and RTS are transmitted for after the data has been sent. Set to 15ms or greater to suit the particular HART device.

### 4.7.7 Spread Spectrum Radio

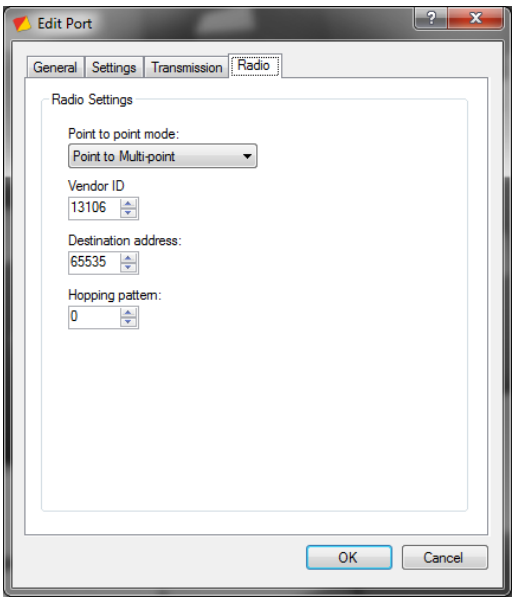
There are three types of Spread Spectrum radio option boards available for the CP-30 and MC-31 modules:

| Country | Option Board | Carrier Frequency | Baud |
|---|---|---|---|
| USA | R4 | 900 MHz | 9600 |
| International | R3 | 2400 MHz | 19200 |
| Australia | R2 | 900 MHz | 9600 |

The basic serial parameters (*Settings* tab) are fixed at the baud rate indicated above. The desired protocol will still need to be selected.

The parameters on the *Transmission* tab are not applicable for the SSR option card.

The settings on the *Radio* tab are as follows:




**Point to point mode:** Selects Point to Point or Point to Multipoint operation.

**Vendor ID:** (16-32767, default=13106) Sets the ID number of the Spread Spectrum radio. All radios on the same network need to have the same Vendor ID in order to communicate with each other. It is recommended that Vendor ID be changed to avoid interference with other radio networks.

**Destination address:** (0-65535, default=65535) Sets the Destination address of the Spread Spectrum radio. All radios on the same network need to have the same Destination Address in order to communicate with each other. It is recommended that Destination Address be changed to avoid interference with other radio networks.

**Hopping pattern:** (0-6, default=0) Sets the Hopping pattern of the Spread Spectrum radio. All radios on the same network need to have the same Hopping pattern to enable them to communicate. To minimize interference from another RTU using a spread spectrum radio, a hopping pattern number that is different to the offending radio should be used.

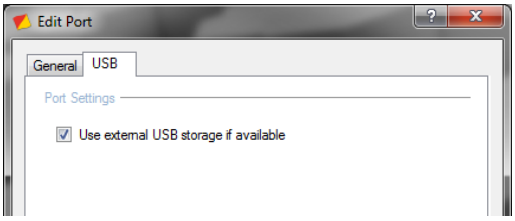
In standard configurations the Vendor ID, Hopping Pattern and Destination Address fields do not need to be modified as the addressing is performed transparently via the RTU addressing system.

It is good practice when implementing a system in a build-up area to not use the default radio settings even if the system is expected to function well. This will help to eliminate interference with other radio devices in the area.

### 4.7.8 USB

A G30 has one fixed USB port that can be used to connect a USB storage device. The G30 is then able to store event logs on the USB storage device. The files can be accessed by manually connecting the USB storage device to a PC.

Note that other types of USB devices such as modems, printers, cameras are not supported.



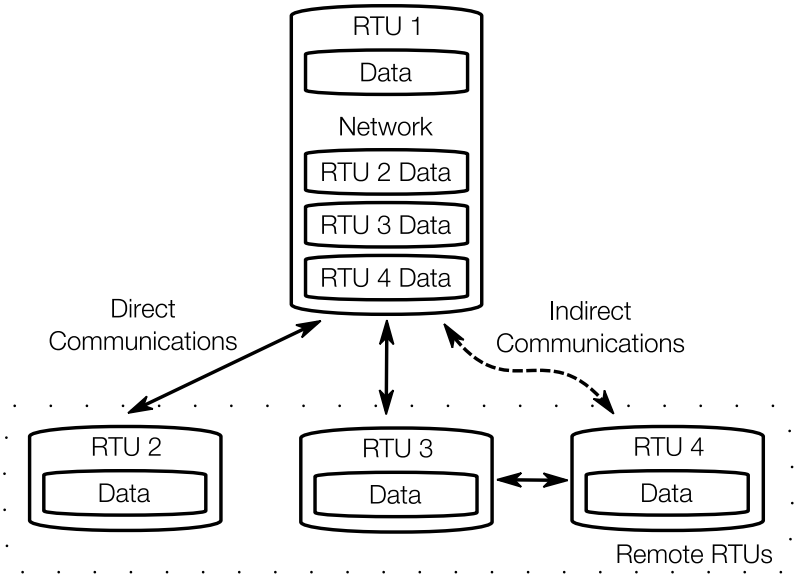

**Use External USB Storage if Available:** When enabled, the G30 will regularly check if a USB storage device is connected to the USB port. If present, event logs will then be saved to the storage device.

## 4.8 RTU Properties – Routes

### 4.8.1 RTU Networking

An RTU network consists of two or more RTUs that can communicate with each other in some way. The communication path is called a **route**.

This example shows RTU1 as the master RTU and RTUs 2-4 as the remote RTUs. RTU3 also stores and then forwards messages between RTU1 and RTU4.



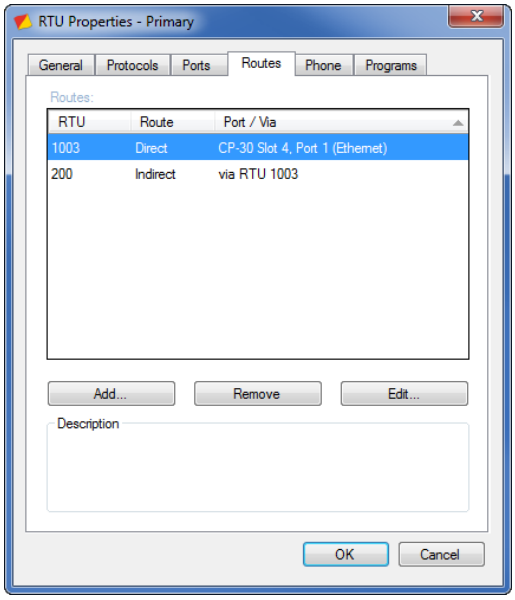

Each RTU has a “route table” that is referred to when communicating with other RTUs or devices in the network. Routes only need to be configured for RTUs that are initiating messages. If an unsolicited message is received from a new RTU, the new RTU route information will be automatically added to the working route table configuration.

In the above scenario, RTU1 is set up to poll the three slave RTUs. It therefore needs to have routes defined which tell it how to access each of the slaves.

In particular, it needs to know:

- the address of the target RTU, which will be used as the “destination” field inside the Kingfisher, Modbus or DNP3 messages
- whether it can send messages directly to the target RTU (a “direct route”), or whether they need to be forward by an intermediary (an “indirect route”). In the above example, all messages for RTU4 need to be sent to RTU3, which will then forward them on, so this would be set up as an indirect route.
- the physical port to use to send the message, which may be Ethernet or serial, and may be on the CP-30 or on an MC-31 communications module
- the physical address to send the message to. For a point-to-point serial connection, this is implicit as the cable can only be connected to one other RTU. For a multi-drop serial connection (e.g. RS485), the physical address is generally the same as the RTU address. For Ethernet, the physical address is the IP address.

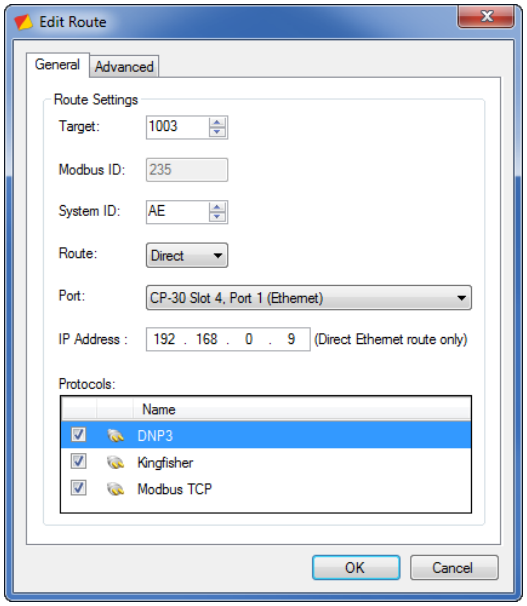
RTU3 will also need a route set to tell it how to reach RTU4.




To configure routes, select the *Routes* tab on the *RTU Properties* dialog.

This shows a summary of all defined routes.

To add a new route, click **Add**; to edit an existing route double click the route, or select it and click **Edit**.




On the *General* tab:

**Target RTU:** (0-65520) The address of the destination RTU to communicate with. Address 0 is only valid for the DNP3 protocol.

**Modbus ID:** When sending a message using the Modbus protocol, only the lower 8 bits of the destination RTU address will be used. This field indicates the equivalent 8-bit Modbus address for the given target address.

**System ID:** (00 to FF Hex, default=AE) This is the communications sync character used at the start of outgoing Kingfisher messages. An RTU will only respond to Kingfisher messages that begin with the same System ID as the RTU's own System ID (as configured in the RTU Properties *General* tab). It is recommended that AE be used for all Kingfisher PLUS+ RTUs except when relaying radio messages as detailed

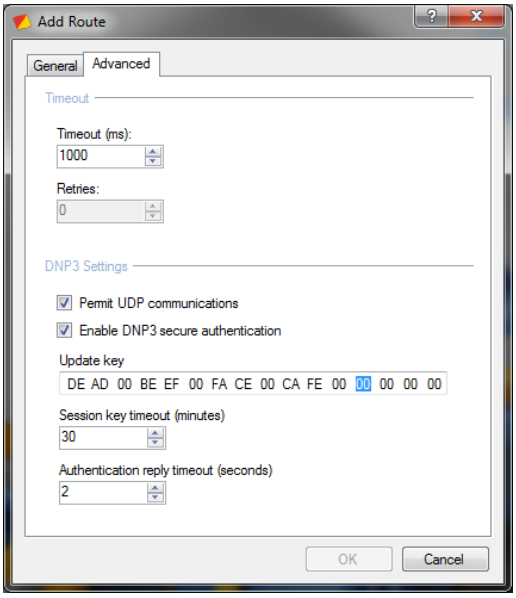
in the topic below Routes - Relaying Radio Messages. System ID is not used by other protocols.

**Route:** (Direct or Indirect) Direct means the RTU is directly connected to the target RTU (e.g. via a private line or radio link). Indirect means the RTU must communicate via one or more other intermediate RTUs to access the target RTU. When an RTU receives a message that is not for itself, it will forward the message to the target RTU if it has a route configured for that target RTU.

**Port / Via RTU:** For a Direct connection, this is the local port number to be used to communicate with the target RTU. For an Indirect connection, this is the directly connected RTU address to which the message must be sent in order to reach the target RTU.

**IP Address:** If the target RTU is connected via Ethernet, its IP address should be set here.

**Protocols:** This specifies the protocol(s) that will use this route. Protocols must first be added to the RTU configuration and then enabled on the port before they can be used on a route (please see the topic RTU Properties - Protocols).




On the *Advanced* tab:

**Timeout:** (0-1000000ms, default 5000ms) The time that an RTU will wait for a reply to its first message. If *Retries* (see below) is set to 1 or greater and a reply is not received, the RTU will send the message again after the timeout has expired.

**Retries: (0-999)** The number of times the RTU will retry sending a message to the target RTU if the previous attempts have failed. The maximum number of attempts is one more than the Retries setting. E.g. if Retries is 3, the RTU will have up to 4 attempts at sending a message.

Retries are generally only useful when using a serial or UDP-based protocol. TCP already includes a retry mechanism so in this case Retries should be set to 0.

In applications where the RTU is regularly polling a slave device, retries should normally only be used if the polling rate is relatively low. For fast poll rates there is no point retrying a poll if the next poll is going to occur within a few seconds anyway.

For DNP3, if retries are required (e.g. because a serial/UDP link is used) then DNP3 Link Layer Confirmations should be enabled. In fact, if the route uses DNP3 then the route Retries setting will be

disabled and forced to 0.

Note: As discussed in the DNP3 Settings section, if you are using the KF_GET_PENDING function block to control the transmission of poll messages, and link layer confirmations are enabled, then the Link layer confirmation timeout should be set to a value which is less than or equal to the Route timeout.

Note: when dialling an RTU, the RTU will dial the primary phone number up to Retries+1 times and then will dial the Secondary phone number up to Retries+1 times until the PSTN modem successfully connects or all the dial retries have failed.

If the route uses DNP3, the following settings are available:

**Permit UDP communications:** By default, DNP3 uses the TCP protocol, which provides error checking and flow control. If this option is enabled then UDP will be used instead. UDP is a connectionless protocol, and is somewhat similar to using a serial link.

Note: This setting will be ignored if the route is assigned to an MC31 Ethernet port. For MC31 ports, only TCP is supported for outgoing connections.

**Enable DNP3 Secure Authentication:** Use this option when the local RTU is a DNP3 Master and is communicating with a DNP3 Slave which requires authentication. When enabled, an Update key can be entered.

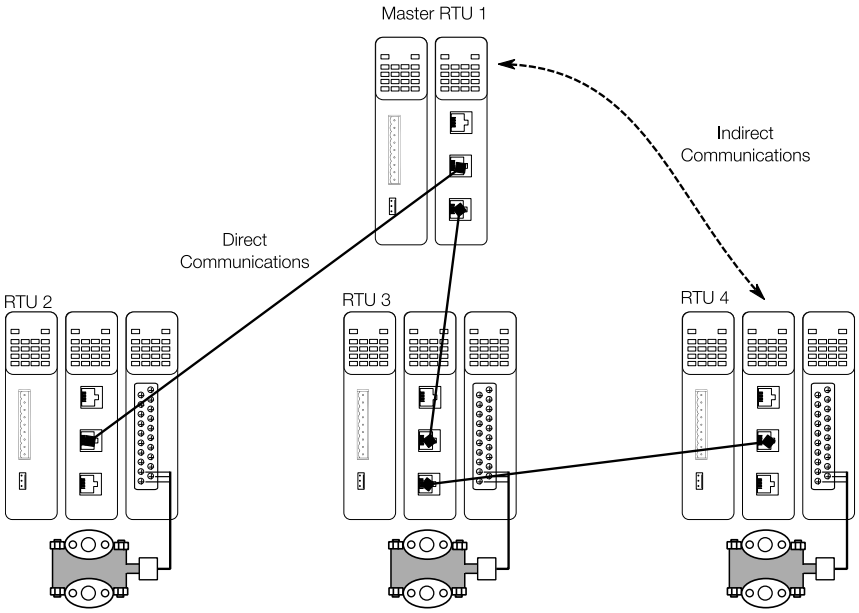
**Session key timeout:** The Update key is used to create an initial session key. The session key is automatically changed each Session Key Timeout interval to protect against “replay” attacks.

**Authentication reply timeout:** How long to wait for a reply to the initial authentication message.

The following sections discuss the route setup for some typical applications.

### 4.8.2 Indirect Routing

In this example, RTU 1 has serial connections to RTU 2 and RTU 3. RTU 3 has a serial connection to RTU 4.



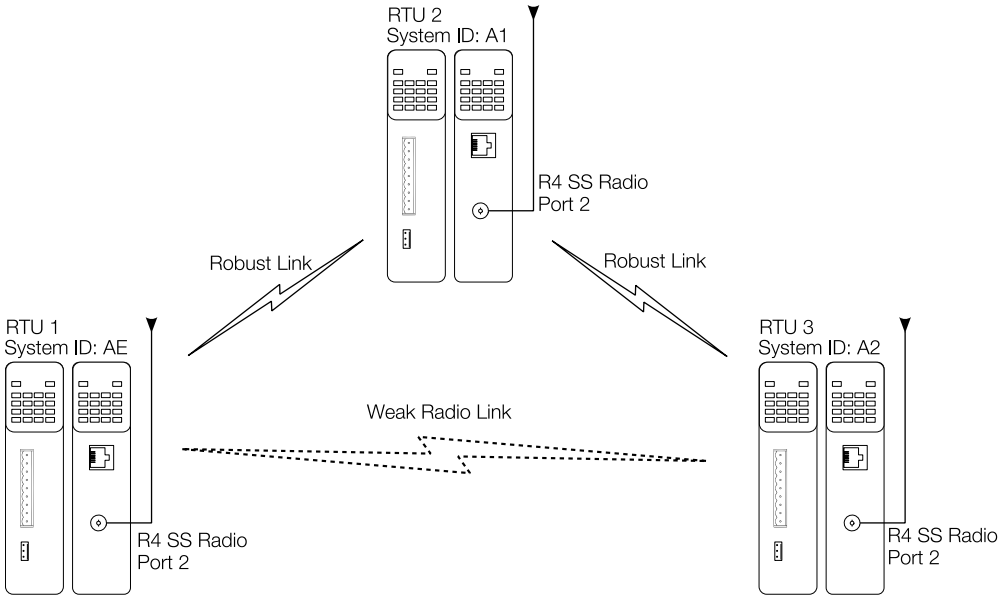

The route configuration described below would allow the master to poll any slave, and any slave to initiate a communication to the master.

| RTU | Route Target | Route |
|---|---|---|
| RTU 1 | RTU 2 | Direct via Port 2 |
| | RTU 3 | Direct via Port 3 |
| | RTU 4 | Indirect via RTU 3 |
| RTU 2 | RTU 1 | Direct via Port 2 |
| RTU 3 | RTU 1 | Direct via Port 2 |
| | RTU 4 | Direct via Port 3 |
| RTU 4 | RTU 3 | Direct via Port 2 |
| | RTU 1 | Indirect via RTU 3 |

If the system is purely master/slave, i.e. RTU 2-4 do not need to initiate communications to the master, then the route configuration could be simplified. RTU1 would still need three routes so it can reach RTU 2, 3 and 4. However the only other route required would be the direct route from RTU 3 to RTU 4.

### 4.8.3 Relaying Radio Messages

For the radio system below, RTU1 communicates with RTU3 by sending a message to RTU2. RTU2 then forwards the message to RTU3. Due to the radio setup, it is sometimes possible for RTU3 to receive the indirect message that is sent to RTU2. This can cause errors since RTU3 will respond to the message at the same time RTU2 is attempting to forward the message.




To prevent both RTU2 and RTU3 responding at the same time, RTU2 and RTU3 are configured with unique system IDs as shown below. RTU1 sends the indirect message to RTU2 with a System ID of A1. RTU3 will only respond to messages with a system ID of **A2** and so ignores the message. When RTU2 forwards the message to RTU3, the message is sent with a System ID of A2. RTU3 then responds to the message.

Note: System IDs 00, AC, A5 and FF are reserved and should not be used.

| RTU | Route Target | Route Sys ID | Route |
|---|---|---|---|
| RTU 1 | RTU 2 | A1 | Direct via Port 2 |
| | RTU 3 | A2 | Indirect via RTU 2 |
| RTU 2 | RTU 1 | AE | Direct via Port 2 |
| | RTU 3 | A2 | Direct via Port 2 |
| RTU 3 | RTU 2 | A1 | Direct via Port 2 |
| | RTU 1 | AE | Indirect via RTU 2 |

### 4.8.4 Dynamic Route Changes

The preceding sections discussed the static configuration of routes in Toolbox PLUS+. Routes can also be added and changed dynamically during operation. There are two main cases:

- When the RTU is operating as a slave, an incoming poll from a master will automatically add or update the route to that master. This is done by recording the port and/or IP address from which the request came.

- The KF_SET_ROUTE function block can be used to explicitly change an existing route – for example to make it use an alternative port. This is often used in a system with redundant communications links, to switch traffic to an alternative link when a communications failure is detected. Note that an existing route to the destination RTU must already exist; this function block cannot be used to create a new route.

The general rule is that a route should be statically configured in Toolbox PLUS+ if the RTU needs to initiate messages to a remote RTU (e.g. it is configured as a master, or it needs to forward messages to another RTU, or it needs to be able to send unsolicited DNP3 messages before any poll messages have been received). If the RTU is acting purely as a slave device then entering a static route to the master is not necessary.

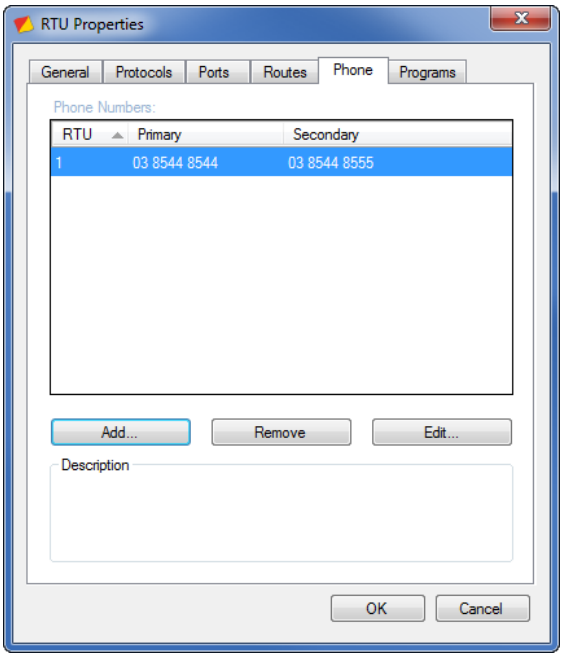
Dynamic route updates do not actually change the RTU configuration. This means that if the RTU restarts then it will revert to its statically configured routes. Also, if you upload the configuration to Toolbox PLUS+ during operation, any dynamic routes will not be visible.

Be aware that a route set dynamically using one of the above two methods may be subsequently overridden by the other method being triggered, which can sometimes cause surprising results.

For example, suppose logic similar to that described in Redundant Communications is used to switch routes when a communications failure is detected. Then if the cable is removed, the route will switch to the alternative port. If the cable is then restored within a minute or two, buffered messages in the remote device may be retransmitted and received by the RTU, which will cause the route to be updated (i.e. restored to the original port).

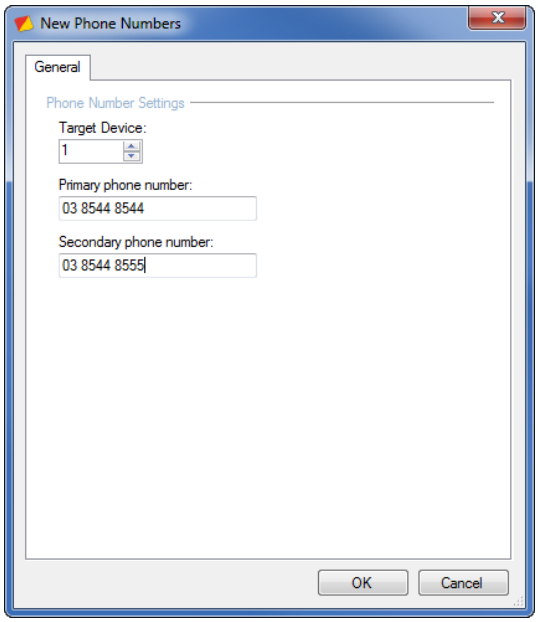
## 4.9 RTU Properties – Phone

If an RTU uses a PSTN port to communicate with remote RTU(s) then the phone number of each remote RTU is configured on the *Phone* tab on the *RTU Properties* dialog.




**Add, Remove or Edit Phone Number**

Select the *Add* button to add a phone number, or select an existing phone number in the list and then select the *Edit* or *Remove* button




**Target RTU:** (1-65520) The address of the destination RTU to dial.

**Primary phone number:** The initial phone number to dial. If connection fails, the RTU will dial the Primary phone number again, up to number of retries configured for the port. Spaces used in the phone number will be ignored by the modem.

**Secondary phone number:** The backup phone number to dial if the Primary phone number fails.

## 4.10 RTU Properties – Programs

One or more programs can be added to the RTU to provide logic processing capability. When editing a program, Toolbox PLUS+ launches the ISaGRAF Workbench editor. Programs or function blocks can also be imported from other RTU configurations or projects.

Programs can be added in the following IEC61131 languages:

**Functional Block Diagram (FBD)** is a graphic language. It allows the programmer to build complex procedures by taking existing functions from the standard library or from the function or function block section.

**Ladder Diagram (LD)** is a graphic representation of Boolean equations, combining Contacts (input arguments) with Coils (output results). The LD language enables the description of tests and modifications of Boolean data by placing graphic symbols into the program chart. LD graphic symbols are organized within the chart exactly as an electric Contact diagram. LD diagrams are connected on the left side and on the right side to vertical Power Rails. Please see the topic ISaGRAF – Logic Examples for Ladder Diagram examples.

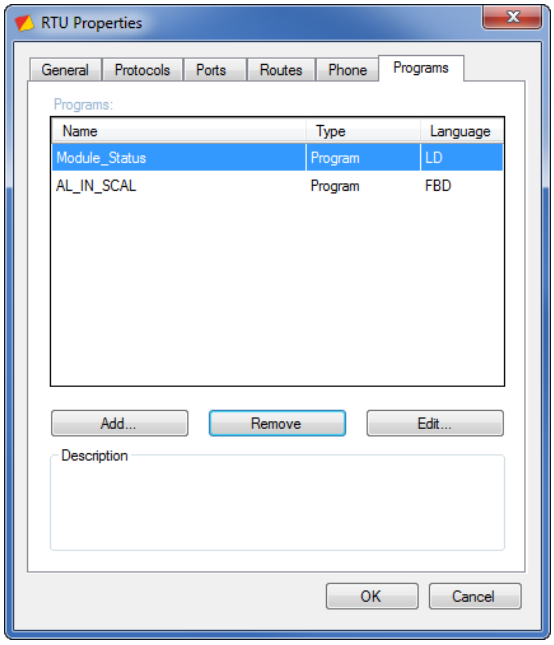
**Sequential Function Chart (SFC)** is a graphic language used to describe sequential operations. The process is represented as a set of well-defined Steps, linked by Transitions. A Boolean Condition is attached to each Transition. A set of Actions are attached to each Step. For programs, Conditions and Actions are detailed using three other languages: ST, IL, or LD. For function blocks, Conditions and Actions are detailed using only two other languages: ST or LD. From Conditions and Actions, any Function or Function Block in any language can be called.

**Structured Text (ST)** is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST language can be used for the description of the actions within the Steps and conditions attached to the Transitions of the SFC or the Actions and Tests of the FC language.

**Flow Chart (FC)** is a graphic language used to describe sequential operations. A Flow Chart diagram is composed of actions and tests. Between actions and test are oriented links representing data flow. Actions and tests can be described with ST, LD or IL programs. Functions and Function blocks of any language (except SFC) can be called from actions and tests. A Flow Chart program can call another Flow Chart program. The called FC program is a sub-program of the calling FC program.

**Instruction List (IL)** is a low level language. Instructions always relate to the current result (or IL register). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

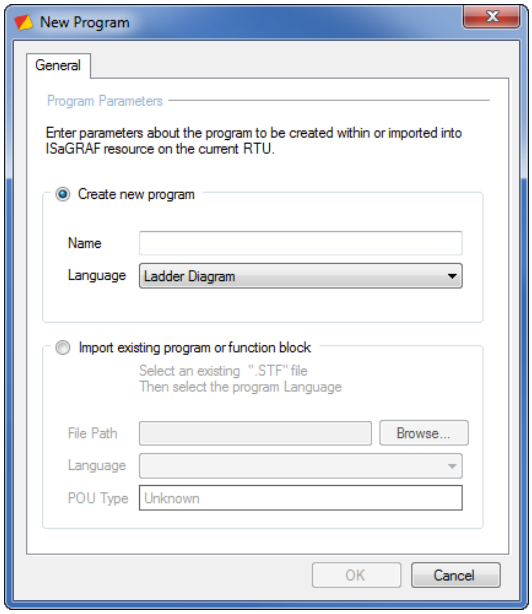
The *Programs* tab lists all currently defined logic programs:




### Add, Remove or Edit Logic Programs

Select the *Add* button to add a program OR select an existing program in the list and then select the *Edit* or *Remove* button.

Pressing *Edit* will open the program in the ISaGRAF workbench.




**Create New Program:** This option is used to create a new program. You need to specify a name for the program and the desired language.

**Import existing program of function block:** If you have an existing ISaGRAF program or function block saved as a ".STF" file then this function allows you to import it into the project.

Note that programs can be created and edited by one of two methods:

- You can use the buttons on *Programs* tab on the *RTU Properties* dialog, as described above. This will launch ISaGRAF and automatically open the editor for the selected program.
- Or you can click the *ISaGRAF* button on the toolbar to run the ISaGRAF Workbench. You can then select the required program from the tree navigator within ISaGRAF, or create a new program.

Any programs created from within ISaGRAF will be reflected in the list in the *RTU Properties* dialog, and vice versa.

## 4.11 RTU Properties – Redundancy

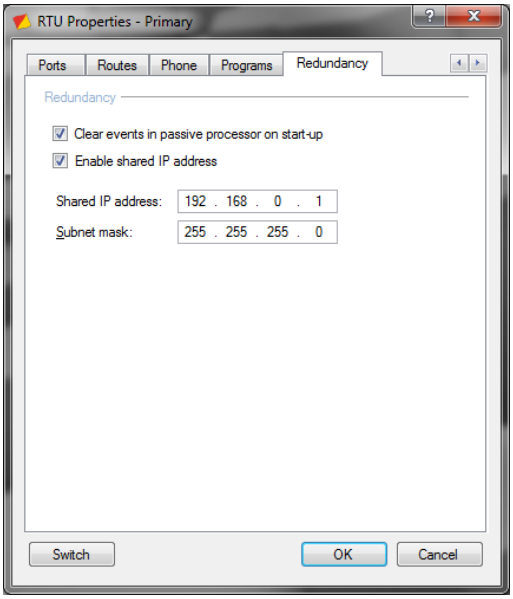
An RTU can have a second CP-30 processor module to provide processor redundancy in case one processor should fail.

As described in Configuring Redundant Processors, redundant CP-30 modules can either use a mirrored configuration (an identical configuration is loaded into the primary and secondary processors), or independent configurations.

If independent configurations are used, the two processors can be configured to share a common IP address. Only the currently active processor responds to messages directed to the shared IP address.

The *Redundancy* tab is enabled if you have configured a redundant processor RTU (two CP-30 modules), and mirror mode is switched off (i.e. the CP-30s each have their own configuration).

To adjust redundancy settings, first ensure that Mirror mode is switched off, then double click on the RTU name in the navigation pane to open the *RTU Properties* dialog. Click on the *Redundancy* tab.




**Clear events in passive processor on start-up:** During normal operation, any events logged by the active processor are transferred to the standby processor. If this option is enabled then any events in the standby processor's memory when it starts up will be cleared, which will ensure that both processors have an identical set of stored events.

**Enable Shared IP Address:** The Shared IP Address allows the SCADA software to communicate with an RTU regardless of which processor module is active.

The Shared IP Address is in addition to each processor's actual IP address, as defined in the Ethernet port settings. Only the currently active processor will respond to messages directed to the shared IP address.

## 4.12 Module Properties

Modules and cards have properties that can be configured after the module or card has been added to the RTU configuration.

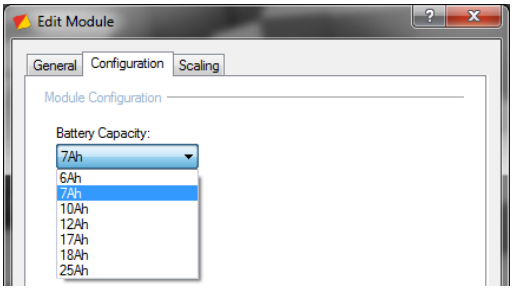
To edit a module's properties, first ensure that the list of modules is displayed (select the *Projects* workspace), then double click on the desired module. This will display the *Edit Module* dialog.

This dialog has a *General* tab, plus other tabs which vary according to module type. The following sections describe the available settings for each module type.

On the *General* tab you can change the module's slot number. If you want to change the type of module then it is necessary to delete the module from the configuration and then add a new one.

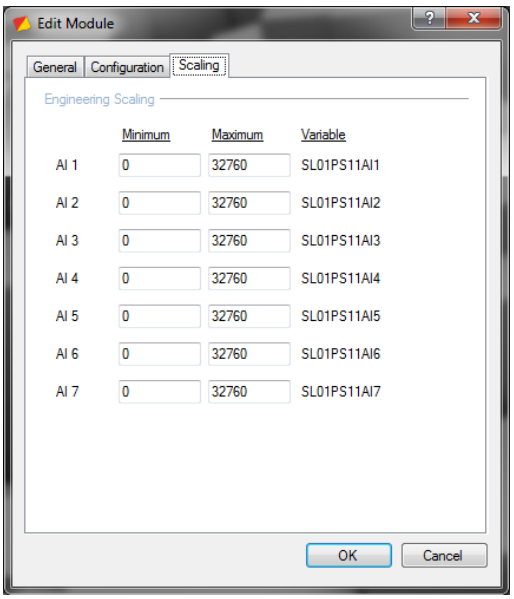
For technical information about the various module types, refer to the *Kingfisher PLUS+ Hardware Reference Manual*, available for download from the CSE-Semaphore support website.

### 4.12.1 PS-1x/2x




On the *Configuration* tab:

**Battery:** The capacity of the installed backup battery.



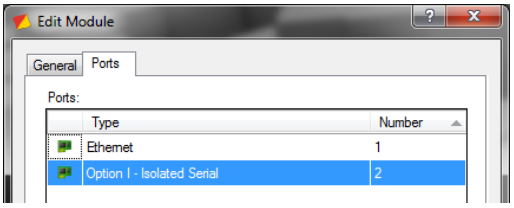

On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

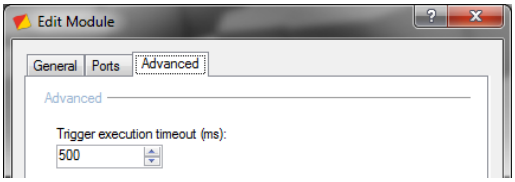
**Maximum:** The desired output value when the raw value is maximum (32760).

### 4.12.2 CP-30




On the *Ports* tab:

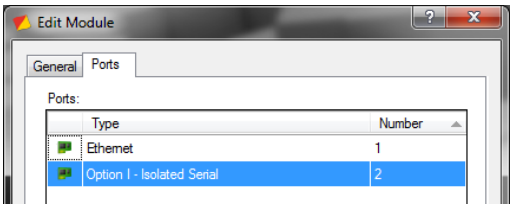
The CP-30's communications ports can be added, removed or edited, as described in RTU Properties - Ports.




On the *Advanced* tab (only visible when the Mirror button is switched OFF):

**Trigger Execution Timeout (ms):** For firmware version 2981 or later this setting is no longer used and will be ignored.

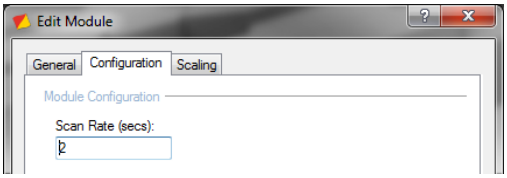
### 4.12.3 MC-31




On the *Ports* tab:

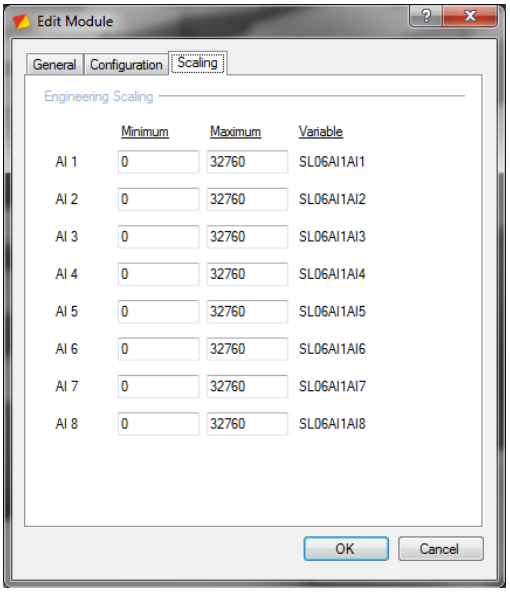
The MC-31's communications ports can be added, removed or edited, as described in RTU Properties - Ports.

### 4.12.4 AI-1/4




On the *Configuration* tab:

**Scan Rate:** How often to scan all the analog input channels



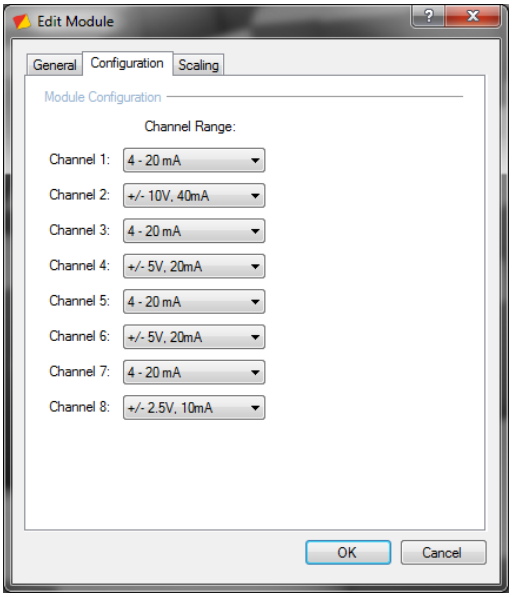

On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

**Maximum:** The desired output value when the raw value is maximum (32760).

### 4.12.5 AI-10



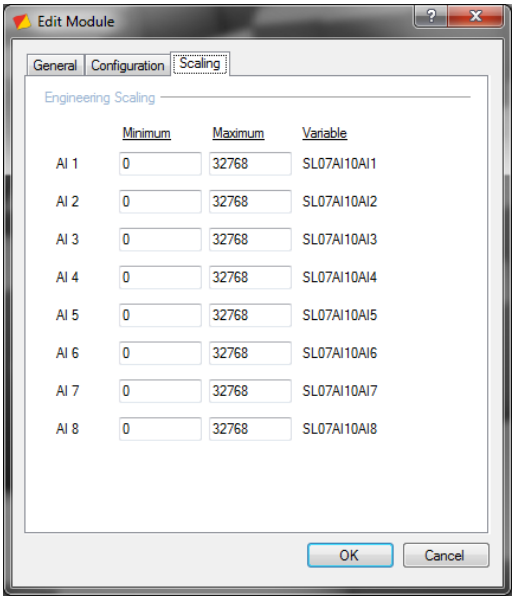

On the *Configuration* tab:

**Channel Range:** The current or voltage input range for that channel. Each channel can be configured individually.

The available ranges are:

- 4 – 20 mA (default)
- 0 – 20 mA
- +/- 40 mA (or +/- 10V for AI-10-V)
- +/- 20 mA (or +/- 5V for AI-10-V)
- +/- 10 mA (or +/- 2.5V for AI-10-V)

For the 0 – 20 mA or 4 – 20 mA ranges, the input will return zero if the current is negative or below 0 or 4 mA respectively.



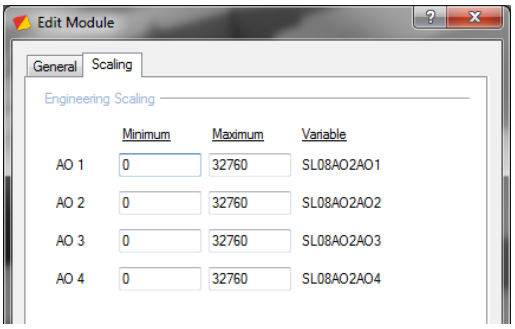

On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

**Maximum:** The desired output value when the raw value is maximum (32760).

### 4.12.6 AI-2/3



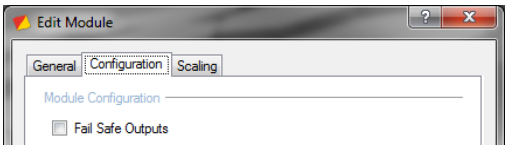

On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

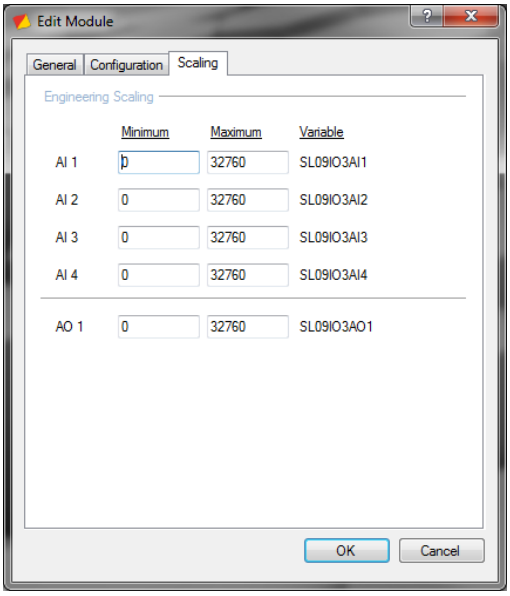
**Maximum:** The desired output value when the raw value is maximum (32760).

### 4.12.7 IO-3/4/5




On the *Configuration* tab:

**Failsafe Outputs:** If enabled and there is no communications activity between the processor and any module on the backplane for 10 seconds the module assumes that the processor has failed and sets the outputs OFF (open). If not enabled (default), all digital outputs will hold their last value.



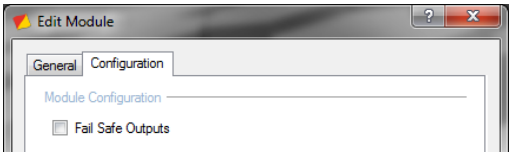

On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

**Maximum:** The desired output value when the raw value is maximum (32760).

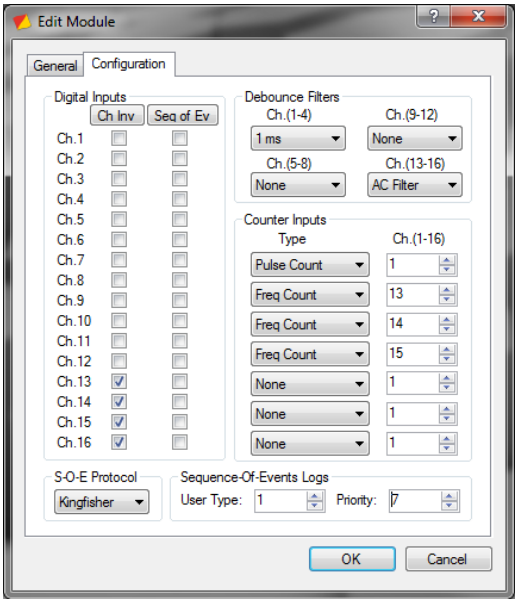
### 4.12.8 DO-1/2/5/6 & IO-2




On the *Configuration* tab:

**Failsafe Outputs:** If enabled and there is no communications activity between the processor and any module on the backplane for 10 seconds the module assumes that the processor has failed and sets the outputs OFF (open). If not enabled (default), all digital outputs will hold their last value.

### 4.12.9 DI-10




On the *Configuration* tab:

**Channel Inversion:** (Tick to enable each channel or select *Ch Inv* button to enable/disable all channels)

By default, when a high voltage level is applied to an input channel, a logical 1 is recorded in the input register and the channel LED is set ON. If the channel inversion option is selected then a high input voltage will result in a logical 0 being recorded in the input register and the channel LED will be set OFF.

**Sequence-of-Events:** (Tick to enable each channel or select *Seq of Ev* button to enable all channels)

When SOE is enabled, any change of state of the input channel (an event) is recorded in the event log to an accuracy of 1 millisecond. Note: The DI-10 has an internal buffer with enough space for 1000 event logs. This means that a DI-10 can cope with bursts of up to 1000 events at a time. Events are uploaded into the processor module at a maximum rate of 100 events per second allowing the DI-10 to cope with events at a sustained rate of 100 events per second. Events are stored in a circular buffer - which causes the oldest event to be overwritten with the newest event when the buffer is full.

**De-bounce Filters:** (None, 1ms, 3ms, 10ms, 30ms, 100ms, 250ms and AC Filter) The value in the digital input register will not update until the input channel has been at the new state continuously for the De-bounce Filter time. This is useful when connecting to mechanical relay or switch contacts. De-bounce filters are applied to groups of 4 channels as shown above. 'AC Filter' is used when connecting AC inputs to the DI-10 module.

**Counter Inputs:** (Frequency, Pulse or Quadrature) The DI-10 can have up to 7 counter inputs which are stored as 16-bit unsigned integer values. Counters can be used on any input channel(s). Note: quadrature counting works on pairs of input channels. Channel pairs are 1&2, 3&4, 5&6, 7&8, 9&10, 11&12, 13&14 and 15&16. So selecting Quad Count on channel 1 will actually work with quadrature on channels 1 and 2. Selecting Quad Count on channel 2 will also work with quadrature on channels 1 and 2, but will reverse the phase of the inputs. The same applies to the other channel pairs used for quadrature inputs.

**S-O-E Protocol:** Specifies whether events should be recorded as Kingfisher or DNP3 protocol events. This is only applicable for inputs where SOE events are enabled.

- If Kingfisher is selected then SOE events (i.e. events with millisecond-accurate timestamps) of the form SL*ss*DI10D*i**n* will be recorded in the RTU event log.
- If DNP3 is selected, then SOE events will be recorded for any DNP3 variables which are mapped to SOE-enabled inputs.

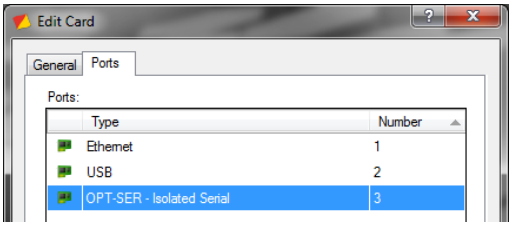
If a DNP3 variable is mapped to a DI-10 input but the input is not SOE-enabled, or the protocol is set to Kingfisher, then “normal” events (generated by the CP30 polling the current state of the input at the end of each logic cycle) will be recorded.

**User Type** (1-31) and **Priority** (0-7): These are applied to event logs generated for channels enabled for Sequence-Of-Events logging. User Type can be used to filter similar types of logs within the event log list. For example SOE logs could be type 1 while analog input logs could be type 2. It will then be possible to only upload type 1 SOE logs or type 2 analog input logs instead of having to upload *all* the event logs together. This setting is not applicable if DNP3 is selected for the SOE protocol.

### 4.12.10 DI-1/5

The DI-1 and DI-5 modules do not require any configuration.

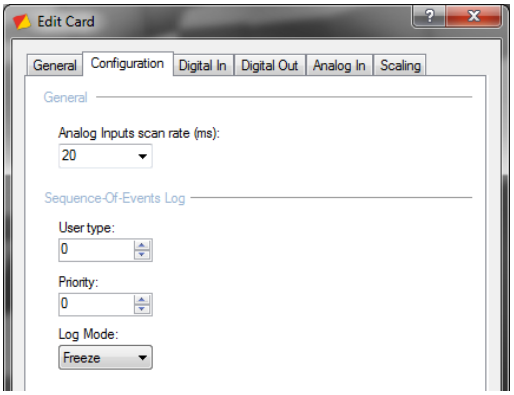
### 4.12.11 G30




On the *Ports* tab:

Port 3 on the G30 can be added, removed or edited, as described in RTU Properties - Ports.

### 4.12.12 IOD-MX2



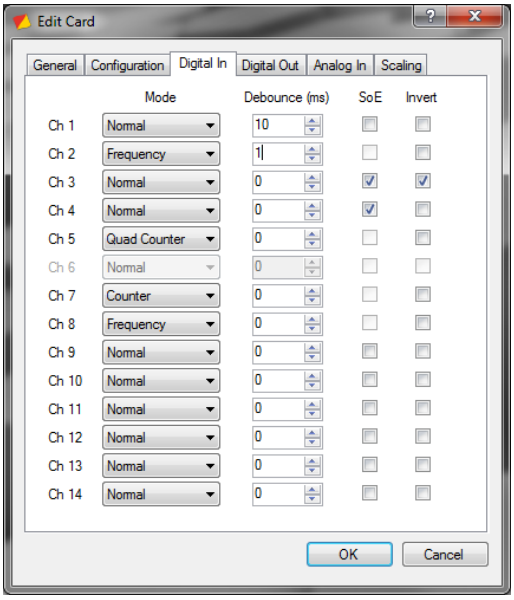

On the *Configuration* tab:

**Analog Inputs scan rate (ms):** How often to scan all the analog inputs.

**User type:** (0-31) The number applied to event logs generated for channels enabled for Sequence-Of-Events logging. User Type can be used to filter similar types of logs within the event log list. E.g. SOE logs could be type 1 while analog input logs could be type 2. It will then be possible to only upload type 1 SOE logs or type 2 analog input logs instead of having to upload all the event logs together.

**Priority:** (0-7) Allows separation of logs within each User Type category.

**Log Mode:** (Freeze, Wrap) *Freeze* stops generating new event logs when the buffer is full. *Wrap* overwrites the oldest event logs when the buffer is full.




On the *Digital In* tab:

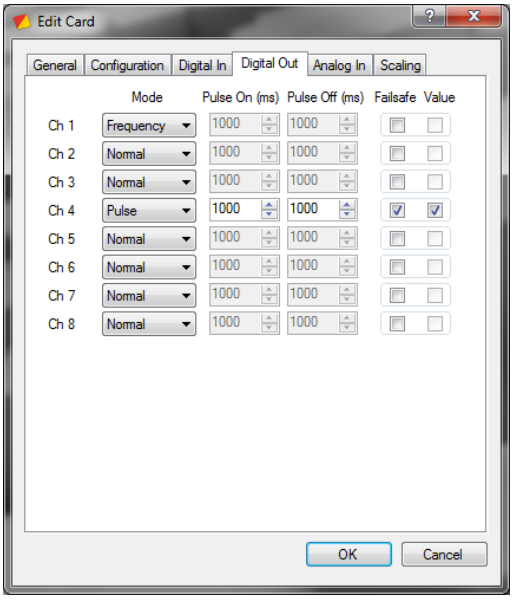
**Mode:** (Normal, Frequency, Counter, Quad Counter) Specifies how to treat the input signal.

- *Normal* sets the input value to 0 or 1.
- *Frequency* counts the number of pulses per second (Hz).
- *Counter* records the total number of rising edges (or falling edges if Invert is enabled).
- *Quad Counter* counts pulses from a pair of channels (3 & 4, 5 & 6, 7 & 8, 9 & 10, 11 & 12, 13 & 14).

**De-bounce:** The digital input value will not update until the input channel has been at the new state continuously for the De-bounce time.

**SoE:** When enabled and Mode is set to Normal, generates an event log whenever the input value changes state.

**Invert:** By default, when the input channel is energised, the value is set to 1. When the channel is inverted and the input channel is energised, the value is set to 0. When mode is set to Counter and invert is set, falling edges will be counted. When Mode is set to Quad Counter and invert is set, the phase of the inputs will be reversed.




On the *Digital Out* tab:

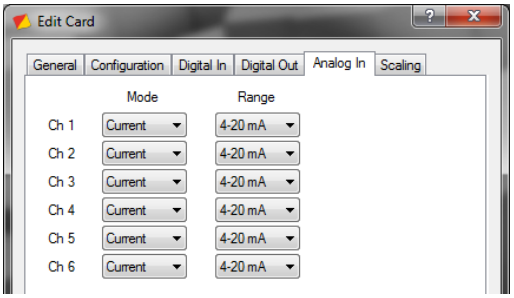
**Mode:** (Normal, Frequency, Pulse) The output channel mode:

- *Normal* sets the channel to open or closed according to variable value (0=open, ≥1=closed).
- *Frequency* outputs a 50% duty cycle signal with a frequency (Hz) given by the variable value.
- *Pulse* outputs the number of pulses given by the variable's value.

**Pulse On / Pulse Off:** The pulse On and Off times to be used when outputting pulses in Pulse mode.

**Failsafe:** If enabled and the IO card loses communications with the G30, the output will be set to the configured value (as detailed below). If failsafe is not enabled (default), the output will hold last state.

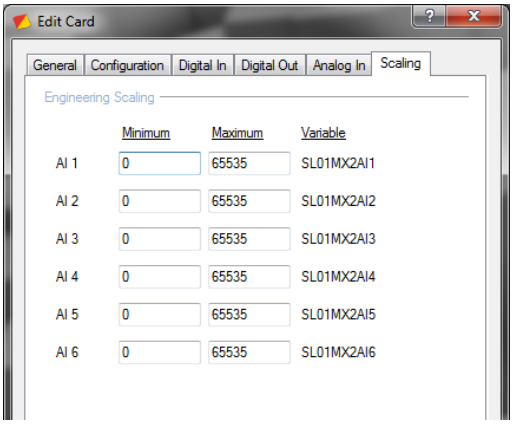
**Value:** The output state to set if failsafe is enabled and triggered.




On the *Analog In* tab:

**Mode:** Selects current or voltage measurement.

**Range:** (0-20 mA, 4-20 mA, 0-5V, 1-5V) The available range settings change according to the Mode setting.




On the *Scaling* tab:

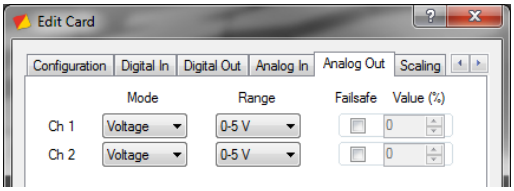
The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

**Minimum:** The desired output value when the raw value is 0.

**Maximum:** The desired output value when the raw value is maximum (32760).

### 4.12.13 IOD-MX3

The IOD-MX3 has the same configuration windows as the IOD-MX2. It also has an analog output configuration window as shown below.




On the *Analog Out* tab:

**Mode:** Selects current or voltage measurement.

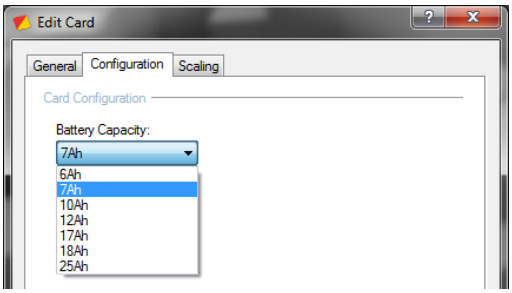
**Range:** (0-20 mA, 4-20 mA, 0-5V, 1-5V) The available range settings change according to the Mode setting.

**Failsafe:** If enabled and the IO card loses communications with the G30, the output will be set to the configured value (0-100%). If failsafe is not enabled (default), the output will hold last value.

### 4.12.14 IOD-MX4

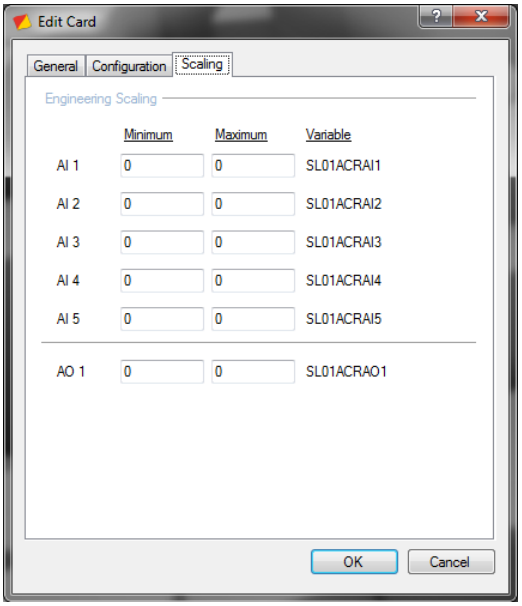
The IOD-MX4 has the same configuration windows as the IOD-MX2, except that the analog input settings are not present.

### 4.12.15 PSO-ACR/DCU




On the *Configuration* tab:

**Battery:** The capacity of the installed backup battery.




On the *Scaling* tab:

The minimum and maximum set-points allow an analog output (with a raw range of 0-32760) to be scaled to a defined range. The scaled value will be stored in the analog output variable.

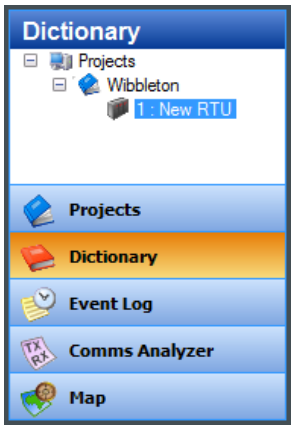
**Minimum:** The desired output value when the raw value is 0.

**Maximum:** The desired output value when the raw value is maximum (32760).

# 5. Dictionary

## 5.1 Dictionary Variables

The RTU uses **variables** to store and access data and these are kept in the **Dictionary**. These variables are also used by ISaGRAF Workbench logic programs. Variables are also sometimes referred to as **symbols**.




**View RTU Variables**

Select the RTU name in the navigation pane

Select *Dictionary* in the Stacked Menu Bar.

When the Dictionary workspace is selected, new variables can be created and existing variables can be edited or deleted.

If an I/O module is added to an RTU, Toolbox PLUS+ automatically adds variables to the dictionary for all the data that are available from that I/O module. For example, if you add a DI-10 module to the project then 23 variables will be automatically created for that module's 16 digital inputs and 7 counters. See RTU Variables for details on the variables associated with each module type.

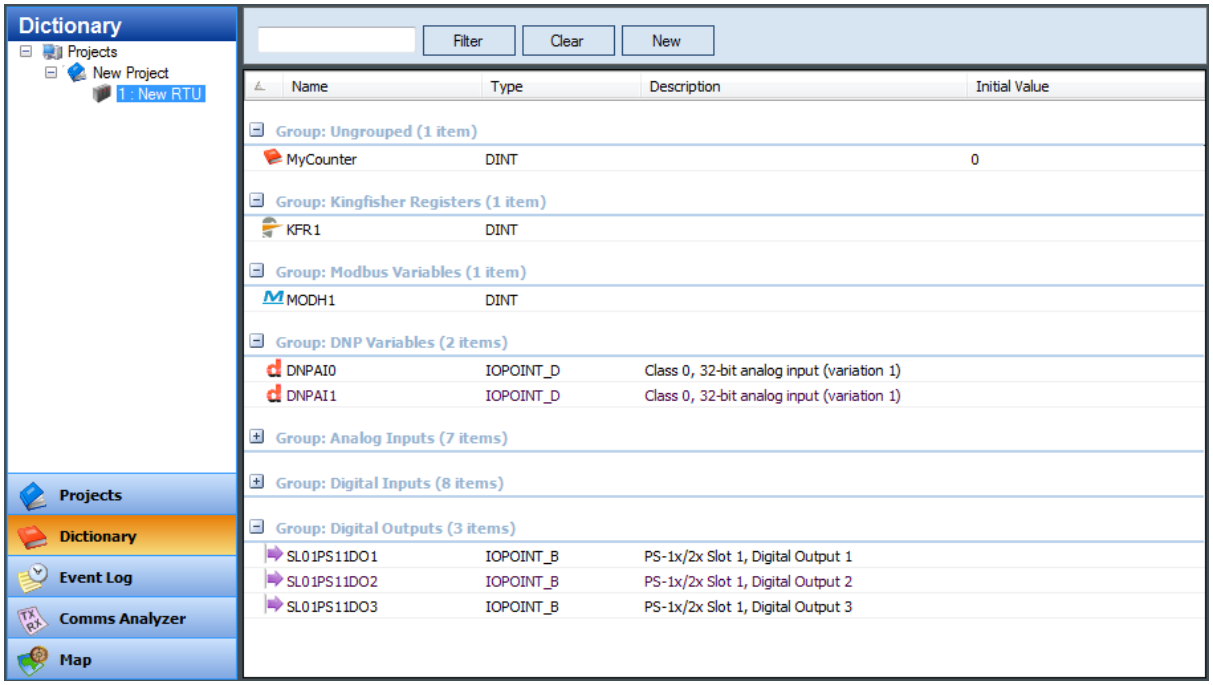
If desired, some or all of these automatically created variables may be deleted if not required. As noted in ISaGRAF Licensing Details, your ISaGRAF licence may be limited to a certain number of symbols (variables), so it may be necessary to delete unused variables in order to fit your application under the variable limit.

### 5.1.1 ISaGRAF

The Dictionary is shared between Toolbox PLUS+ and ISaGRAF Workbench. All variables automatically or manually created in Toolbox PLUS+ will appear in the ISaGRAF dictionary list (press in ISaGRAF Workbench) and may be used in logic programs. Likewise, all global variables created in ISaGRAF will appear in the Toolbox PLUS+ dictionary.

### 5.1.2 Dictionary Workspace

An example Dictionary view for an RTU with various types of variables is shown below.




As can be seen in the above example, the Dictionary variables are grouped into categories, or groups. This process is largely automatic. The '+' and '-' buttons can be used to expand and collapse each group.

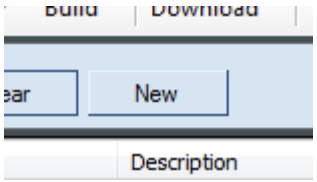
For each variable, the following information is shown:

- Variable name. Automatically generated variables have particular naming conventions, e.g. DNPAI1 is DNP analog input register #1, and SL01PS11DO3 is digital output #3 on the PS11 module in slot 1. User defined variables (e.g. MyCounter) can have any name. (Note that the name "PS11" is used for all PS-1x/2x module types.)
- Variable Type. Each variable has a particular data type. For example: 16- bit integer (INT), 32-bit integer (DINT). Some variables have a compound type, e.g. IOPOINT_B, which as well as the actual value (a Boolean value in this case) also contains other details such as a timestamp and data quality flags.
- The Description is simply a free text description of the variable. For automatically generated variables a suitable description is also generated.
- Initial value. For user defined variables an initial value can also be specified.

The Dictionary workspace includes some features to make it easier to deal with what can be very long lists of variables:

- **Filter** button. The Dictionary can be filtered so that only variables that contain the characters specified in the filter are displayed. Enter a few characters from the name you are looking for and click *Filter*. Press *Clear* to clear the filter and show all variables.
- **Sort**. By default, variables are sorted by group. By clicking on the column headings the list can be sorted by the contents of those columns.

## 5.2 New Dictionary Variables




**New Variable**

Select the RTU name in the navigation pane

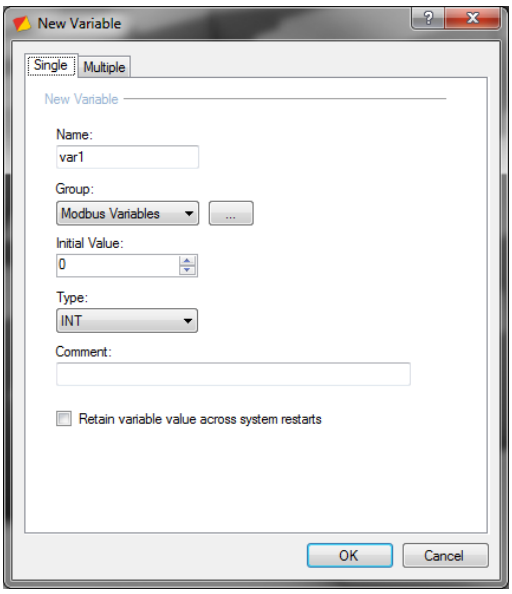
Select *Dictionary* in the Stacked Menu Bar.

Select the *New* button (located above the variable list). The *New Variable* dialog will be displayed. This allows you to create either a single variable, or a block of automatically generated variables.

### 5.2.1 Creating a Single Variable

The *Single* tab on the *New Variable* dialog is used to create a single variable.

Note: If the variable is to be mapped to a protocol (e.g. Kingfisher, Modbus, DNP3) it is better to use the *Multiple* tab (see below), as this enforces the required naming convention for these types of variable.




**Name:** (1-127 characters) The variable name. There are certain requirements on variable names:

- Variable names must start with a letter.
- Only the first 16 characters of variable names are event logged.
- Variable names can contain the letters **a** to **z** or **A** to **Z** (lower or upper case), the digits **0** to **9** or the underscore character **_**. Variable names cannot contain spaces. For example, **Pump5_Running** is OK, while **Pump#5 Running** is invalid because it contains a # character and spaces.
- Variables starting with the underscore character are reserved for use by Semaphore and ISaGRAF.
- Reserved ISaGRAF keywords (e.g. “FALSE”) cannot be used as variable names (see ISaGRAF - Reserved Variable Names for a complete list).
- When a Kingfisher, Modbus or DNP3 variable is created, it will be displayed with the corresponding symbol. User variables that are not recognized by Toolbox PLUS+ are displayed with the symbol.

**Group:** The name of the Dictionary group to display the variable in. To create a new group name, select the “…” button.

**Initial Value:** If specified, this is the initial value used by the logic program after the configuration is downloaded, after a restart or after a power reset. The value of a variable can be saved during a restart by enabling “Retain variable across system restarts” (see

below). The format of values that can be used for each variable type is detailed in the ISaGRAF help.

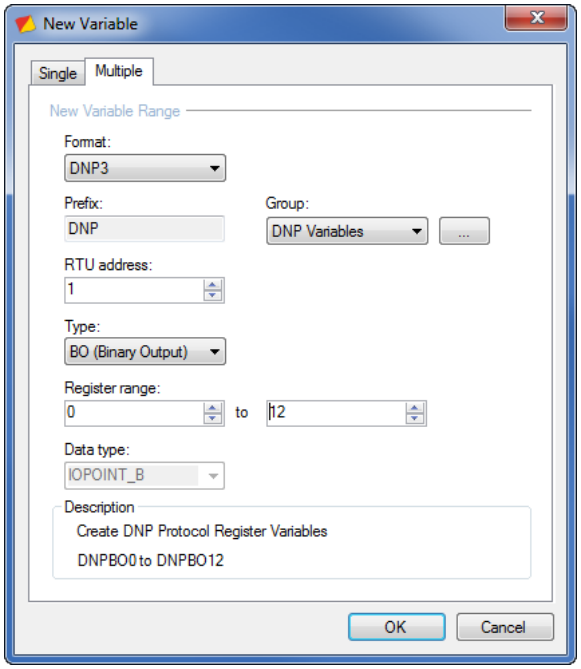
**Type:** The data type of the variable. ISaGRAF supports a wide range of data types as detailed in the topic ISaGRAF – Variable Types.

**Comment: (0-128 characters)** Description of the variable.

**Retain variable value across system restarts:** When enabled, the variable's last known value is saved and restored after an RTU restart or power failure. The last known value will not be saved and restored after downloading firmware, logic or a configuration. After a download, the variable's Initial Value will be used.

### 5.2.2 Creating Multiple Variables

A range of variables can be automatically created by selecting the *Multiple* tab. As the various fields are set, the names of the variables that will be created are displayed at the bottom of the dialog. The following parameters can be specified:




**Format:** If *DNP3*, *Modbus* or *Kingfisher* is selected, the variable prefix will be set to DNP, MOD or KF respectively and the *Type* field will be appropriately constrained to suit the protocol.

**Prefix:** The initial characters of the variable names. This can only be modified if *Format* is set to *Free*.

**Group:** The name of the Dictionary group to display the variable in. To create a new group, select the "…" button.

**RTU Address:** (1-65520) The source of the variables. Use the local RTU address to create registers for locally generated data. Use the address of a remote RTU to create variables for data received from that remote RTU.

**Type:** The characters to insert between the RTU address and the register number. This will vary depending on the selected format:

- DNP Types: AI (Analog Input), AO (Analog Output), BC (Binary Counter), FC (Frozen Counter), BI (Binary Input) and BO (Binary Output).
- Modbus Types: C (Coil), D (Discrete), H (Holding) and I (Input).
- Kingfisher Types: R (Local Register) and F (Floating Point Register – used with HART protocol only)
- Free format: Any string can be entered here

**Register Range:** The register numbers to use for the variables. For Modbus and Kingfisher formats, registers are numbered from 1. For DNP3, registers are numbered from 0.

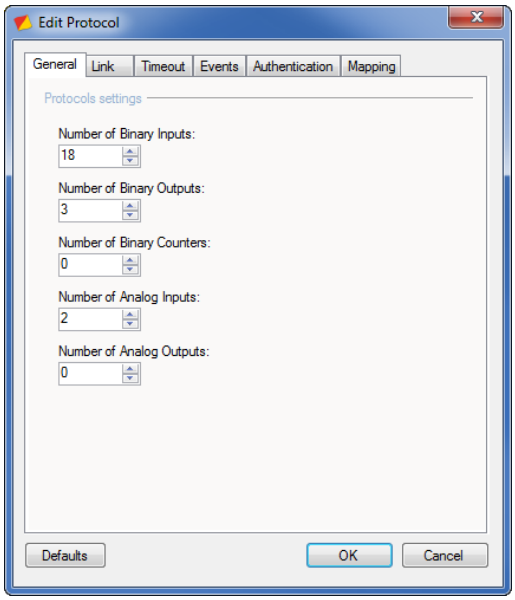
**Data Type:** This will be set automatically according to the selected *Type* value. For free format variables, a data type can be selected from the list. See ISaGRAF – Variable Types.

The Description at the bottom of the window shows the range of variables that will be created when OK is selected.

### 5.2.3 Creating DNP3 Variables

DNP3 variables created using the *New* button, as described above, will have default settings (Class 0, variation 1). If you wish to change the class or variation of the new variables then you can do so by double clicking on each variable individually, as described in Editing Variables. However this can be tedious if you have more than a few variables.

An alternative method of creating arrays of DNP3 variables with non-default settings is as follows:

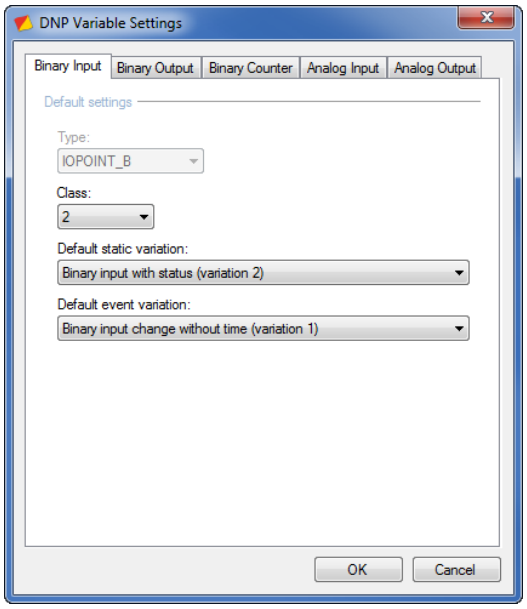



Open the DNP3 *Edit Protocol* dialog, as described in DNP3 Settings. This dialog will show the number of currently defined DNP3 variables of each type.

Adjust each of these controls to the desired number of variables.

Note: If you reduce these numbers then existing variables will be deleted.

Click *Defaults* to open the *DNP Variable Settings* dialog.



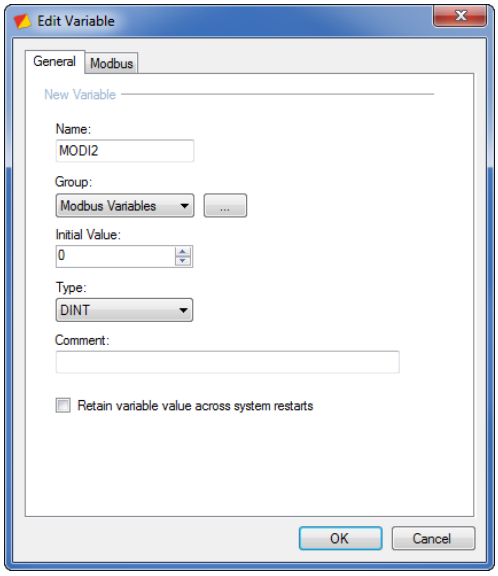

On each tab, select the desired class, variation and any other settings (e.g. high/low limits and dead band for analog inputs) See Editing Variables for more details.

After pressing OK to close all dialogs, the required variables will be created.

## 5.3 Editing Variables

To delete a variable, select it in the Dictionary workspace and press the DEL key (or right click and select *Delete*).

To edit a variable, double click on it (or right click and select *Edit*). This will open the *Edit Variable* dialog:




The *General* tab contains settings common to all variable types:

**Name:** Name of the variable. If the variable is to be used for data transfer using a communications protocol (Kingfisher, Modbus, DNP3, etc.) then it must follow the appropriate naming conventions, as described in RTU Variables.

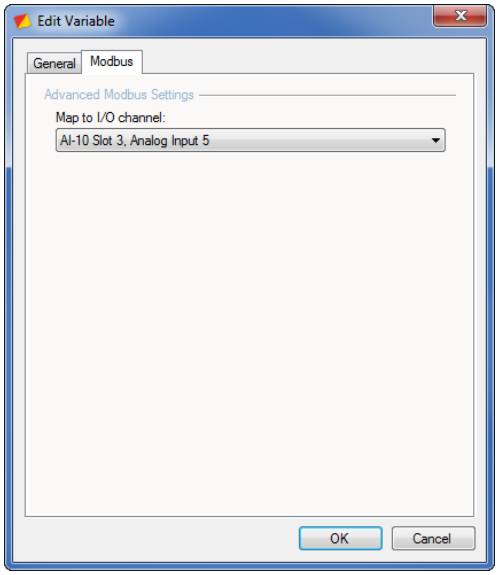
**Group:** Variable groups are purely an aid to keeping a project organised – the group to which a variable is assigned does not affect its function. I/O module and protocol variables will be automatically assigned to appropriate groups. The "…" button can be used to create new group names.

**Initial value:** The initial value of the variable. If not specified then 0/FALSE will be used.

**Type:** The data type for I/O module and protocol variables cannot be changed. For general variables, any type can be set here.

**Comment:** Free text field, for documenting the variable's function.

**Retain value:** If this option is checked then the variable value will be saved to non-volatile memory after each logic cycle. If the RTU restarts for any reason then the last value will be restored when logic execution resumes. Note that if this option is enabled then an initial value cannot be set.

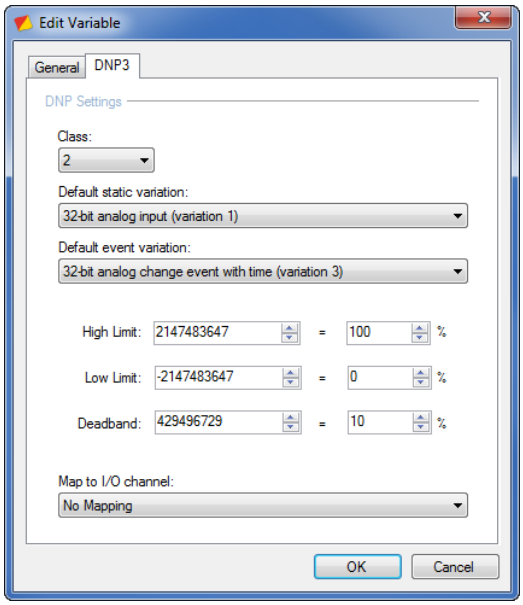



For Modbus variables:

**Map to I/O channel:** This allows an I/O channel to be mapped to a Modbus variable. Only those I/O channels whose type is appropriate to the type of Modbus variable will be available for selection.

For example, a MODD*n* (discrete input) Modbus variable can only be mapped to digital input channels, and only to channels which are not already mapped to another variable.

For analog input channels, the value stored in the variable will be the scaled value, i.e after applying the I/O module's defined scaling factors.




For DNP3 variables:

**Class:** DNP3 event class (None, 1, 2, 3). If set to *None* (class 0) then events (changes in value) will not be logged for this variable. The current value of the variable will still be reported.

Classes 1-3 are arbitrary categories to which variables and the events they generate can be assigned. These classes can then be polled at different rates (for example).

**Default static variation:** Specifies the data format to use when reporting or requesting this variable's current value.

**Default event variation:** Specifies the data format to use when reporting or requesting historical events for this variable. Not applicable if Class is set to *None*.

**Map to I/O channel:** This allows an I/O channel to be mapped to a DNP3 variable.

For DNP3 analog input variables:

**High Limit:** If the variable value is above this limit (specified either as an absolute value or a percentage of the maximum value) then an Over Range value will be reported.

**Low Limit:** If the variable value is below this limit then an Under Range value will be reported.

**Deadband:** An event will only be recorded if the variable's value changes by at least this amount. Not applicable if Class is set to *None*.

## 5.4 Exporting and Importing Variables

If you need to create or modify a large number of variables then it may be easier to export the dictionary to an Excel spreadsheet file, edit it in Excel, then re-import it.

To export the dictionary, select the required RTU, then *File* → *Export* → *To Excel…*,

When editing the variables in Excel, be sure not to modify the format. Create new rows by copying existing rows.

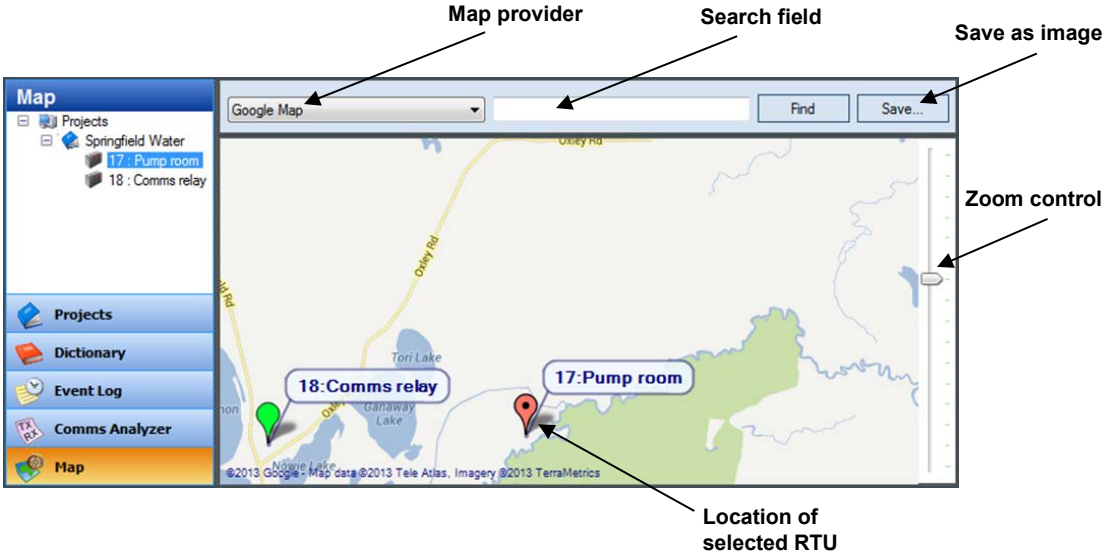
To import the dictionary, select the required RTU, then *File* → *Import* → *From Excel…*,

# 6. Maps

## 6.1 Viewing RTU Locations

The *Map* workspace allows the geographic locations of the defined RTUs to be visualised.

To view the RTU locations, select a project or RTU name in the navigation pane, then click *Map* in the Stacked Menu Bar.




The map can be scrolled by dragging with the mouse, and zoomed using the slider on the right hand edge.

The configured locations of the RTUs in the current project are shown as “push pins” on the map. The pin for the selected RTU is highlighted in red.

By default, the *Google Maps* service is used to retrieve the map data. However, alternative providers can be selected using the control at the top of the workspace. Note that an internet connection is required in order to load map data.

You can also search for place names by entering them in the Search field, and clicking *Find*.

Finally, map displays can be saved (e.g. as a JPG file) by clicking the *Save* button.

## 6.2 Positioning an RTU

**Positioning an RTU on the Map**

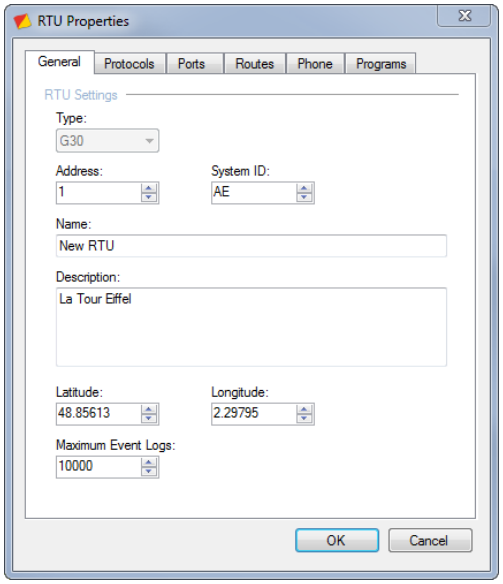
Select the RTU name in the navigation pane

Select *Map* in the Stacked Menu Bar

Enter the address (or closest landmark) of where the new RTU will be located and click the *Find* button

Once the location has been found, right click on the desired location and select *Place RTU Here.*

To move the selected RTU on the map, simply repeat the above procedure.

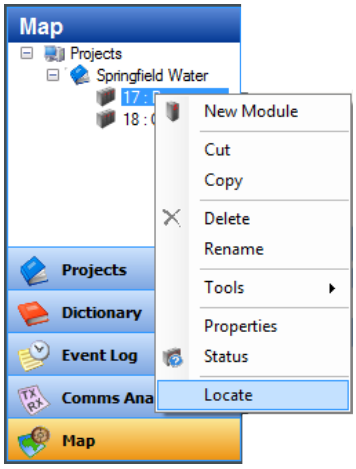



**Manually positioning RTU**

An alternative method is to directly enter the latitude and longitude in decimal degrees (negative for Southern or Western hemispheres) on the *RTU Properties* dialog.

## 6.3 Finding an RTU

Once an RTU's geographical position has been saved in the project, you can then display its position on a map, as follows:




Select the RTU name in the navigation pane

Right click on the RTU and select *Locate.* The *Map* workspace will open and indicate the location of the RTU.

# 7. ISaGRAF

## 7.1 ISaGRAF Overview

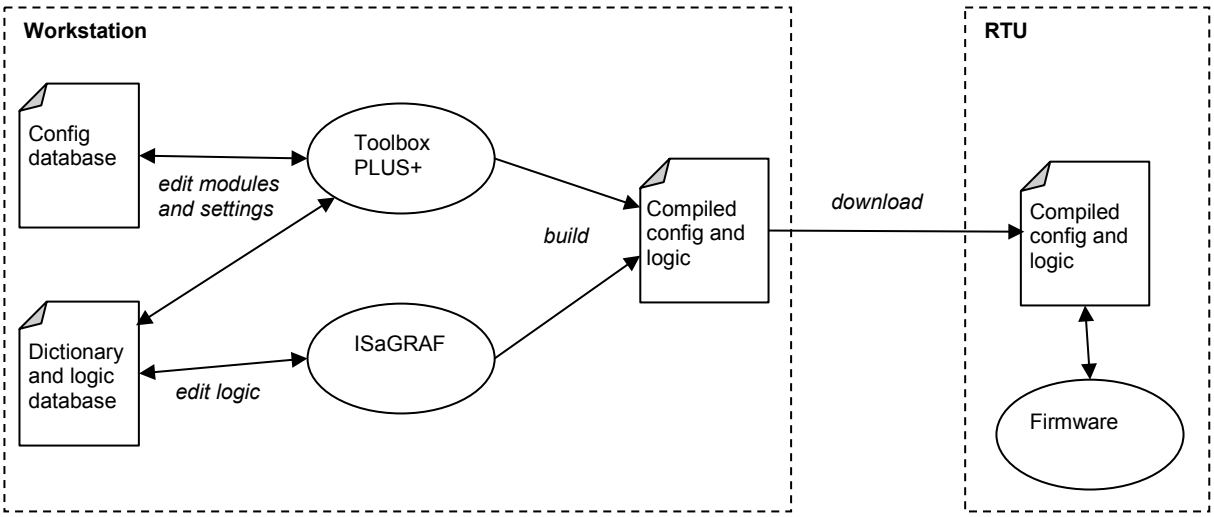
**ISaGRAF™** provides the logic processing for each RTU. ISaGRAF allows logic to be created in any of the IEC 61131 control languages: ladder diagram (LD), structured text (ST), function block diagram (FBD), sequential function chart (SFC) or instruction list (IL). The flow chart (FC) language can also be used. See RTU Properties – Programs for more details about the various languages.

The ISaGRAF editor (called Workbench) is used to create and edit logic programs. As noted in RTU Properties – Programs, ISaGRAF may be launched either by:

- selecting the required RTU, then clicking the *ISaGRAF* toolbar button. The required logic program can then be selected from inside ISaGRAF.
- right-clicking on the required RTU and selecting *Properties*, then selecting the *Programs* tab, then selecting the required program, then clicking *Edit*.

ISaGRAF and Toolbox PLUS+ are separate applications, but are closely integrated. Both share a common database so that variables created in ISaGRAF are visible in Toolbox PLUS+, and vice versa.

The following diagram illustrates how Toolbox PLUS+ interacts with ISaGRAF and the RTU during the process of configuring the RTU.




When you define and configure modules in Toolbox PLUS+, these settings are saved to a configuration database. Variables, whether created automatically when a module was added or manually, are saved to the dictionary, which is part of the ISaGRAF logic database. These two databases are linked by the overall Toolbox PLUS+ project, which may also contain configuration and logic databases for other related RTUs.

Before the configuration settings entered in Toolbox PLUS+ and the logic entered in ISaGRAF can be used by the RTU, they need to be “compiled” or “built”. This can be done by clicking the *Build* toolbar button in Toolbox PLUS+, or it can be done automatically when a configuration is downloaded to the RTU.

The final step is to download the compiled configuration and logic to the RTU, which is done by clicking the *Download* toolbar button in Toolbox PLUS+. You will be prompted to rebuild the configuration and/or logic if Toolbox PLUS+ detects that they have changed since the last time they were built. Once the download is complete the RTU will restart and the firmware will then begin processing the newly loaded configuration and logic.

Note that while ISaGRAF is running, most functions in Toolbox PLUS+ are disabled. These will be re-enabled once all ISaGRAF windows are closed.

## 7.2 Function Blocks

Most of the work in a logic program is carried out by **function blocks**. Programming consists of selecting the appropriate function blocks and tying them together using the desired language, e.g. ladder diagram, structured text, etc. Most function blocks have input parameters which specify the data to operate on and/or option settings.

ISaGRAF provides many built in function blocks, which are divided into a number of categories, e.g. arithmetic, process control, signal generation, logical operations, and so on. All are documented in the ISaGRAF Workbench on line help. You can also create user-defined function blocks.

A number of custom function blocks have been developed to support the operation of Kingfisher RTUs. In ISaGRAF, these are listed in the “Kingfisher” category. Most of these function blocks relate to communications protocols, event logging or system parameters. Kingfisher function blocks are documented in this manual – see ISaGRAF Function Blocks.

The section ISaGRAF - Logic Examples includes a number of practical logic fragments for performing real world tasks.

## 7.3 Getting Started

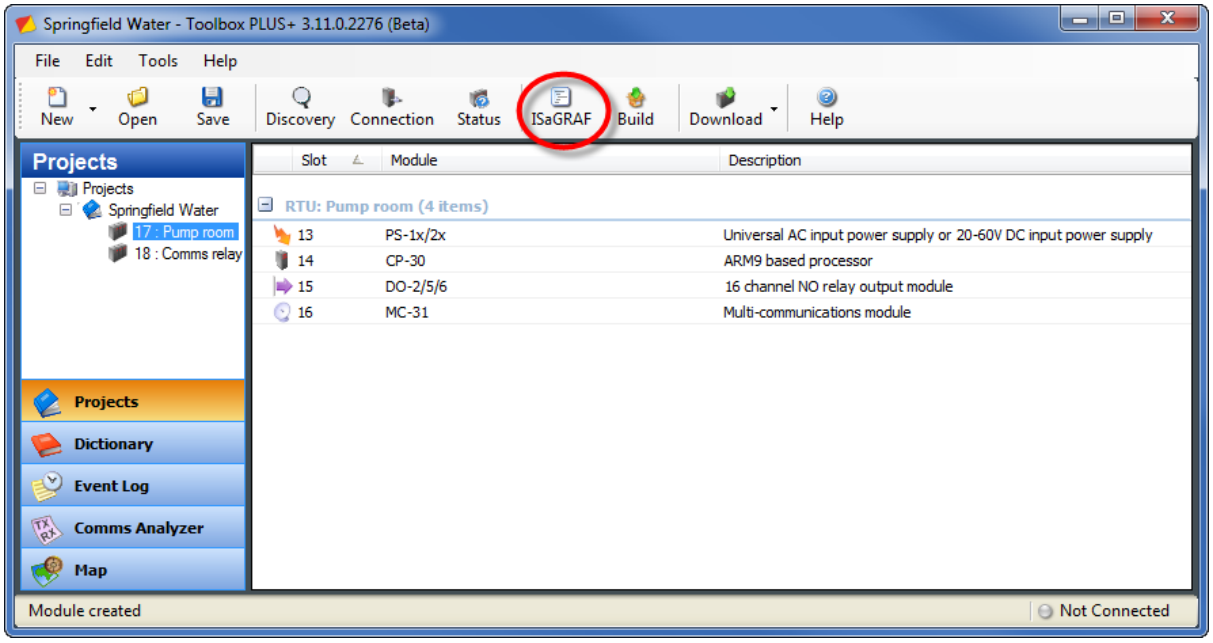
This section provides a brief tutorial on how to enter, compile and run a simple logic program.

Note: Some familiarity with Ladder Diagram programming is assumed in this tutorial.

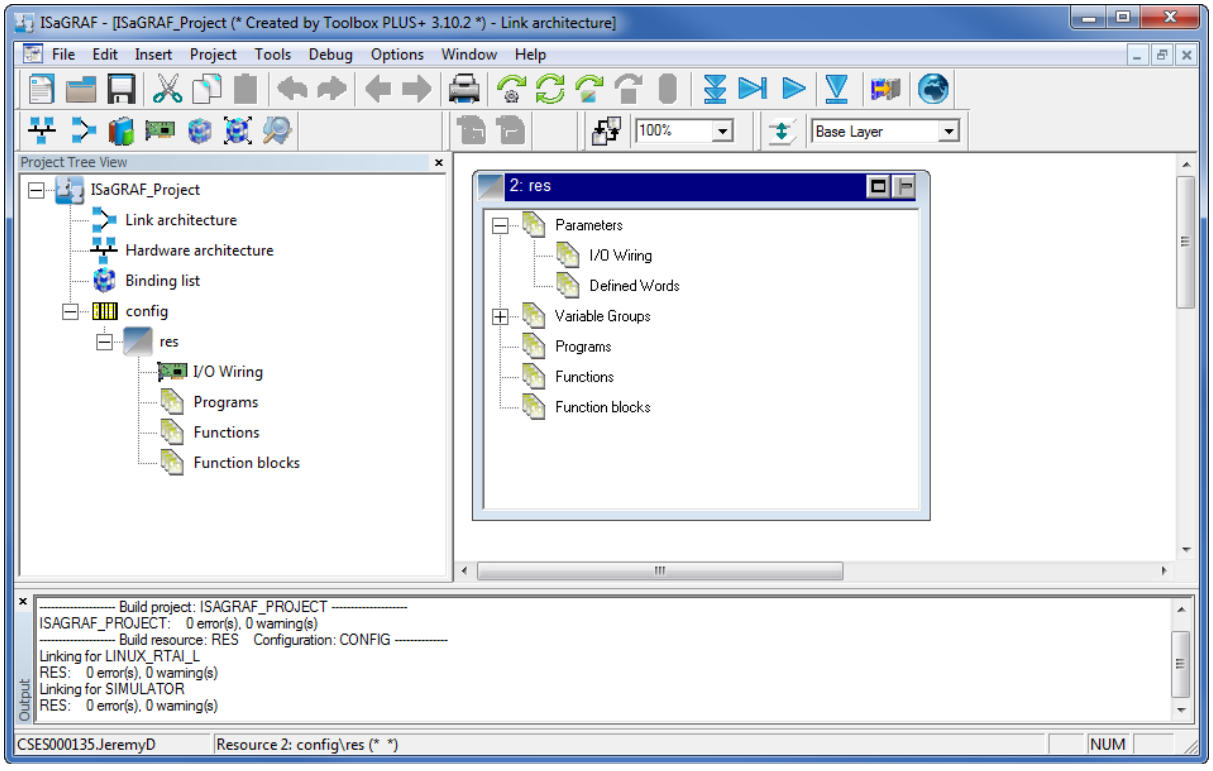
### 7.3.1 Defining a Program

The first step is to define the modules that make up the RTU. In this example, the 4-slot RTU described in RTU Configuration - Example is used. Once the RTU has been defined, ensure that the correct RTU is selected in the navigation pane, then press the *ISaGRAF* toolbar button to start ISaGRAF.

Note: If you have an ISaGRAF licence dongle, ensure that it is plugged into a USB port on your computer before starting ISaGRAF. See ISaGRAF Licensing Details for more information.




The ISaGRAF Workbench will now load. By default, this will show the *Link Architecture* screen.

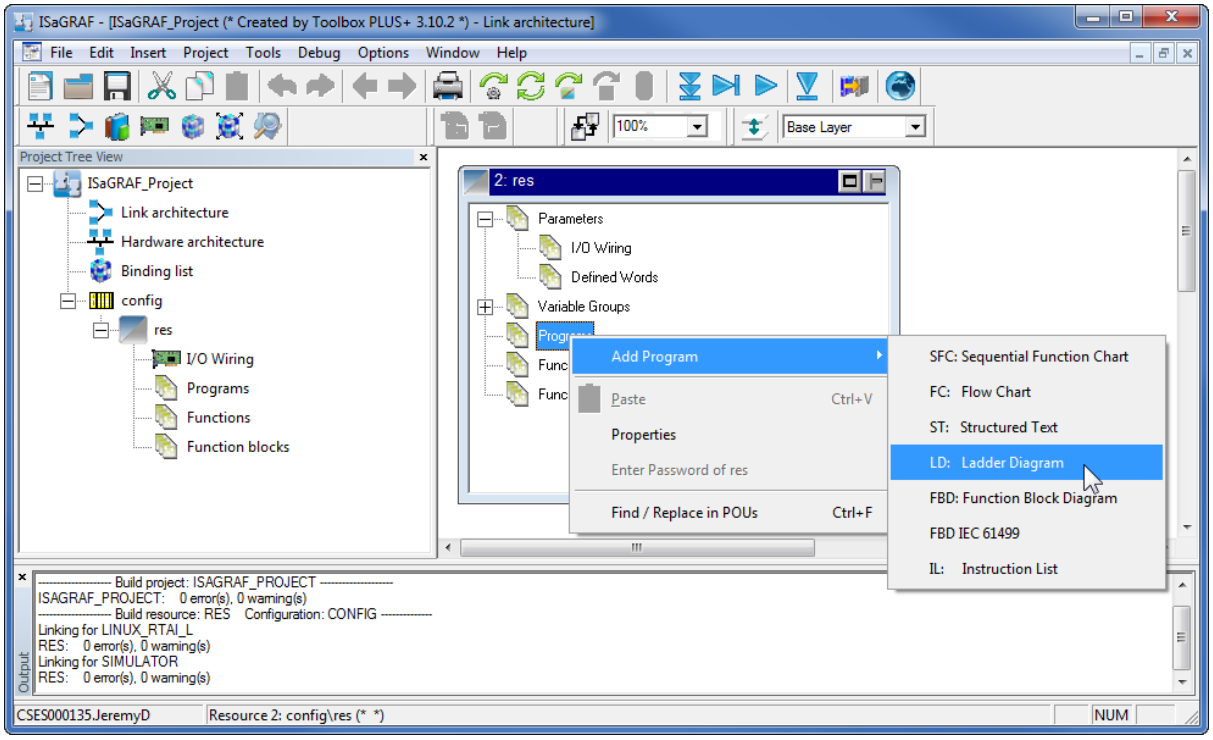



While ISaGRAF provides its own project-related features (represented by the tree view on the left), these are not used in a Kingfisher RTU environment. Projects (collections of RTUs) are instead managed by Toolbox PLUS+. Each RTU defined within a Toolbox PLUS+ project is associated with its own independent ISaGRAF project.

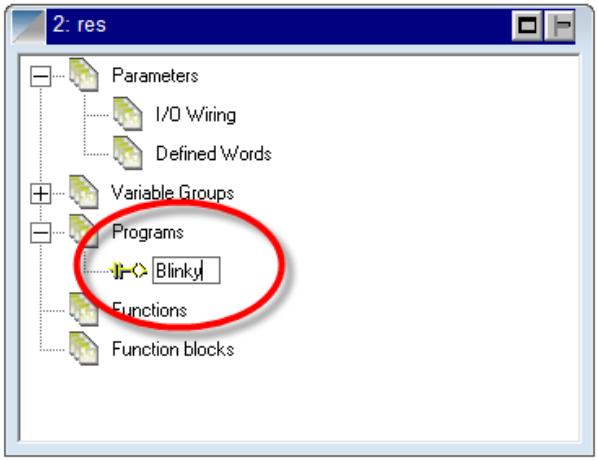
In ISaGRAF, something that can execute logic is called a **resource**. In a Kingfisher system, the ISaGRAF project contains exactly one resource: the RTU itself. The window labelled *res*

lists the logic programs and function blocks (collectively referred to as **POUs**, or program organisation units) which have been defined for the RTU.

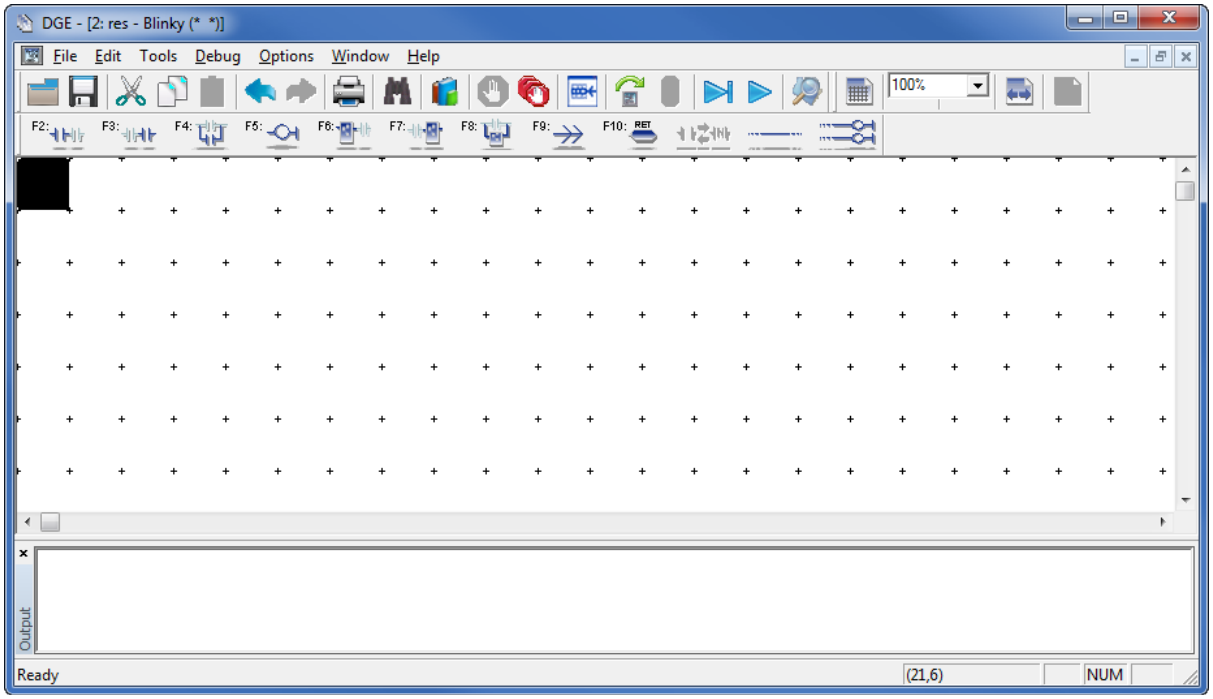
In this example, no logic has been defined yet. We will now create a new program using the Ladder Diagram language. Right click on *Programs*, then select *Add Program* and *LD: Ladder Diagram*.




Enter a name for the program:




Now double click on the program name. The appropriate editor for the selected language (Ladder Diagram in this case) will then open:




### 7.3.2 Entering Logic

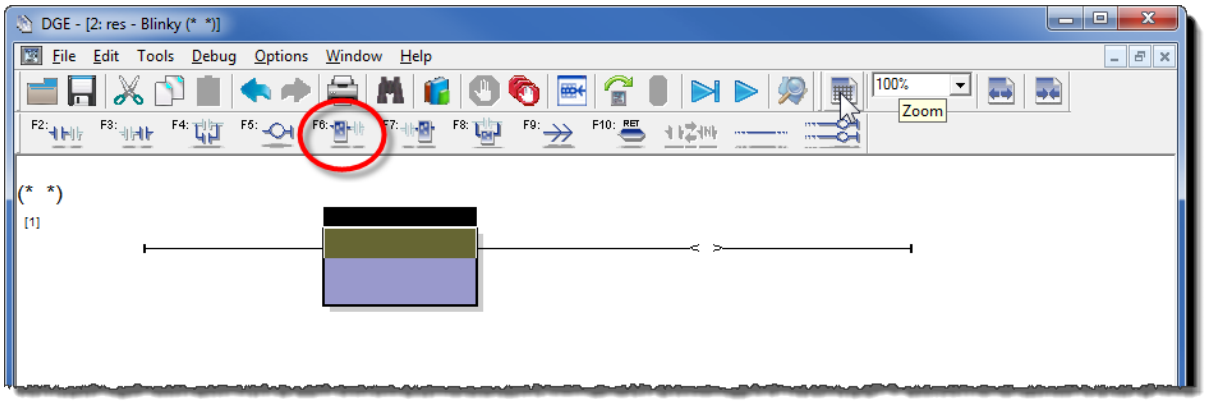
In this example, the following display settings have been changed to improve legibility:

- Switch off the grid (*Options* menu > *Layout*, then uncheck *Display grid*)
- Expand the spacing horizontally (click button twice)

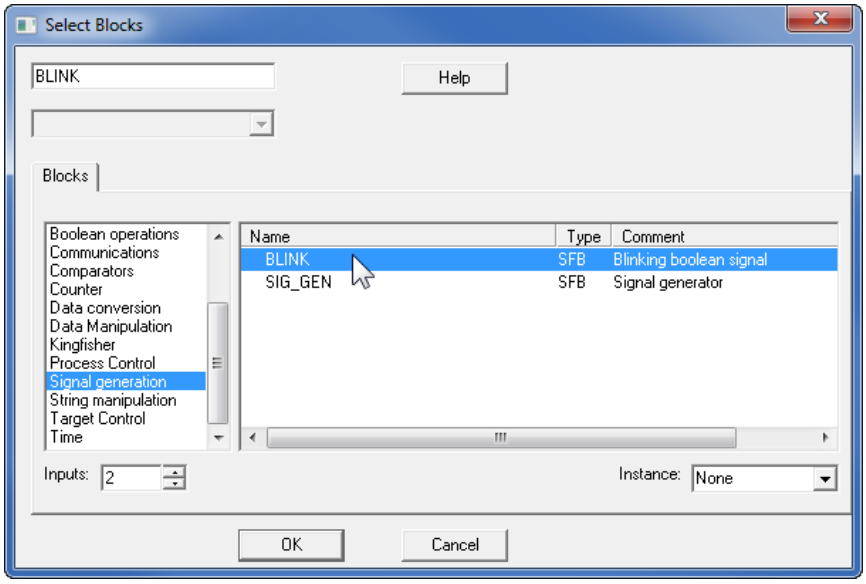
Most functions in the Ladder Diagram editor can be performed using toolbar buttons. The bottom row contains buttons to insert ladder diagram elements such as contacts (Boolean inputs), coils (Boolean outputs) and function blocks.

In this simple example, we will create a single rung of logic which will flash one of the LEDs on the digital output module. A series contact will allow the flashing to be stopped.

Begin by clicking the toolbar button to insert a function block. Because nothing is selected, a new logic rung will be created, consisting of an unnamed function block and coil. (In Ladder Diagram, every rung must contain an output, or “coil”.)



Double click on the function block to select the required block. In this case, the one we want is called *BLINK*, which is in the *Signal generation* category. Select the category, then the function block name, then click *OK*. If you want to learn more about this function block, click *Help*.



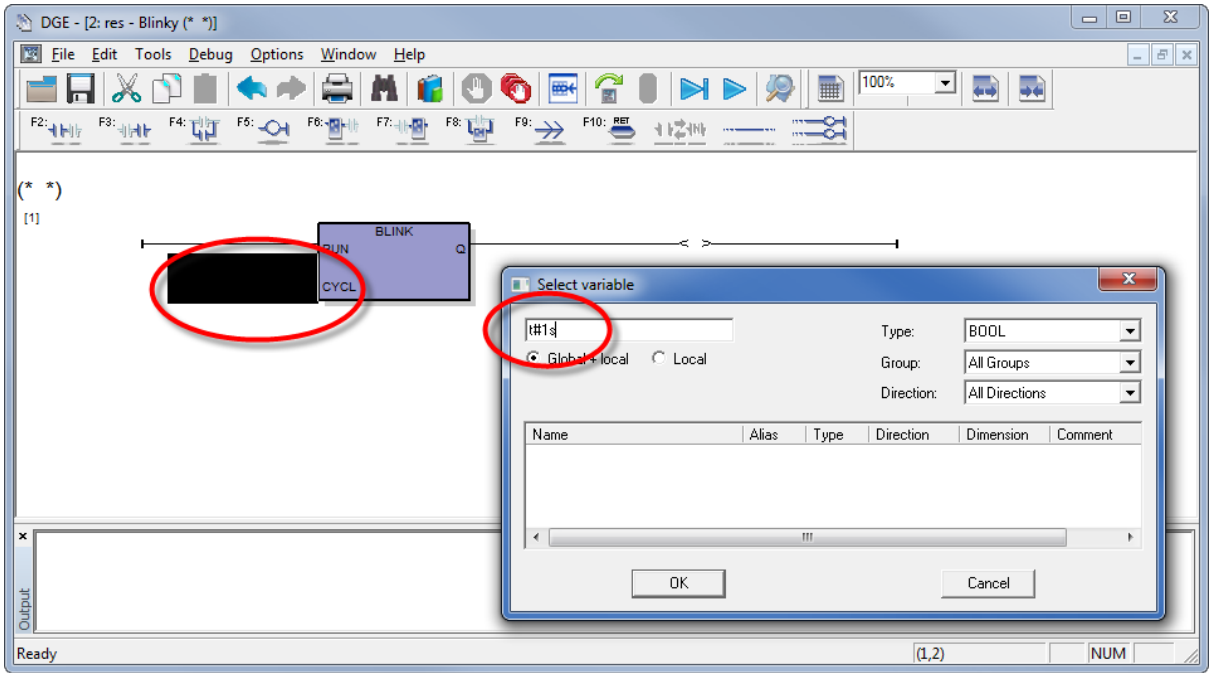

The ladder diagram will now be updated to contain the name of the function block.

The *BLINK* function block has one output and two inputs:

- *Q* is a Boolean output which will switch between true and false at the configured rate.
- *RUN* is a Boolean input. If true, then the *Q* output will oscillate as described above.
- *CYCL* is a time input, which sets the oscillation period.

We now need to enter the desired time period. Double click the space to the left of the *CYCL* label to display the *Select variable* dialog. Here you can enter a constant value, create a new variable or select an existing variable.

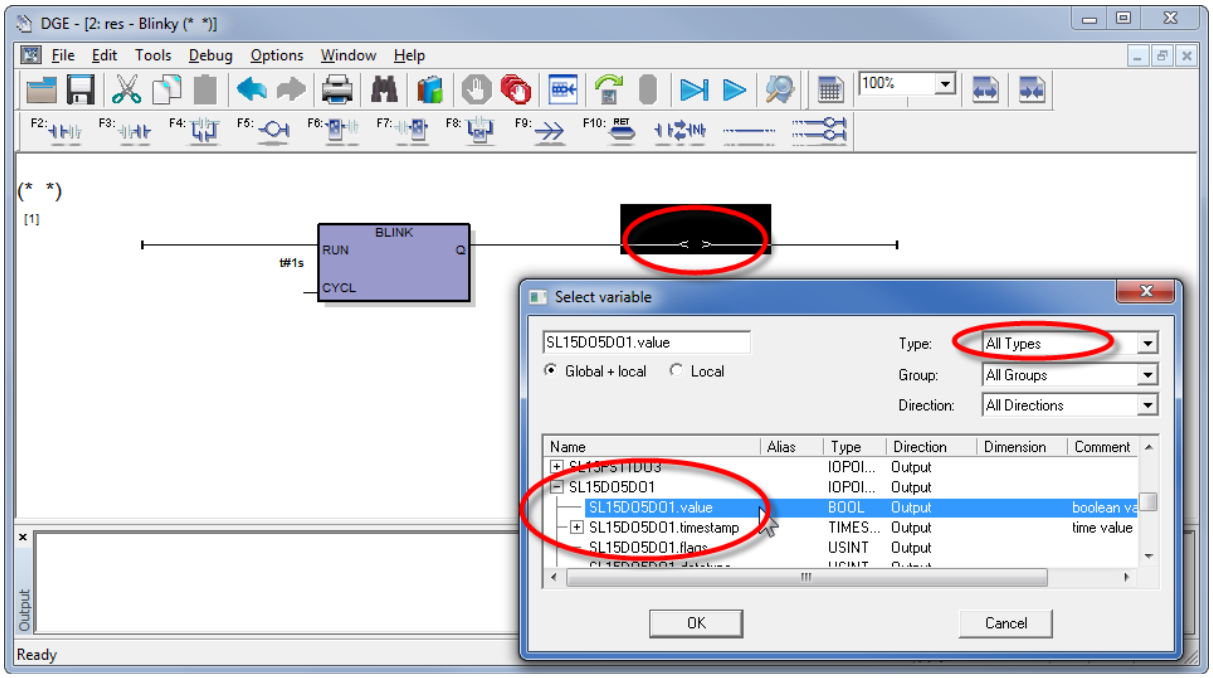
In this case the time period is constant, so just enter *t#1s* then click *OK*. Time interval constants consist of *t#,* followed by an integer, followed by a unit (*ms, s, m, h* or *d*).




Now we need to associate the coil with an output (say output #1) on the digital I/O module in slot 15. Double click on the coil to again bring up the *Select variable* dialog.

ISaGRAF variables for each of the module's output points were automatically created by Toolbox PLUS+ when the module was added. Note however that I/O points are not just simple Boolean variables; they are compound structures which contain timestamp and flag values in addition to the actual data value.

First ensure that the *Type* field is set to *All Types*. Then scroll down the list of variables to find *SL15DO5DO1* ("slot 15, DO-5 module, digital output 1"). Click the "+" icon to expand the sub-fields within the variable, then select *SL15DO5DO1.value* and press *OK*.



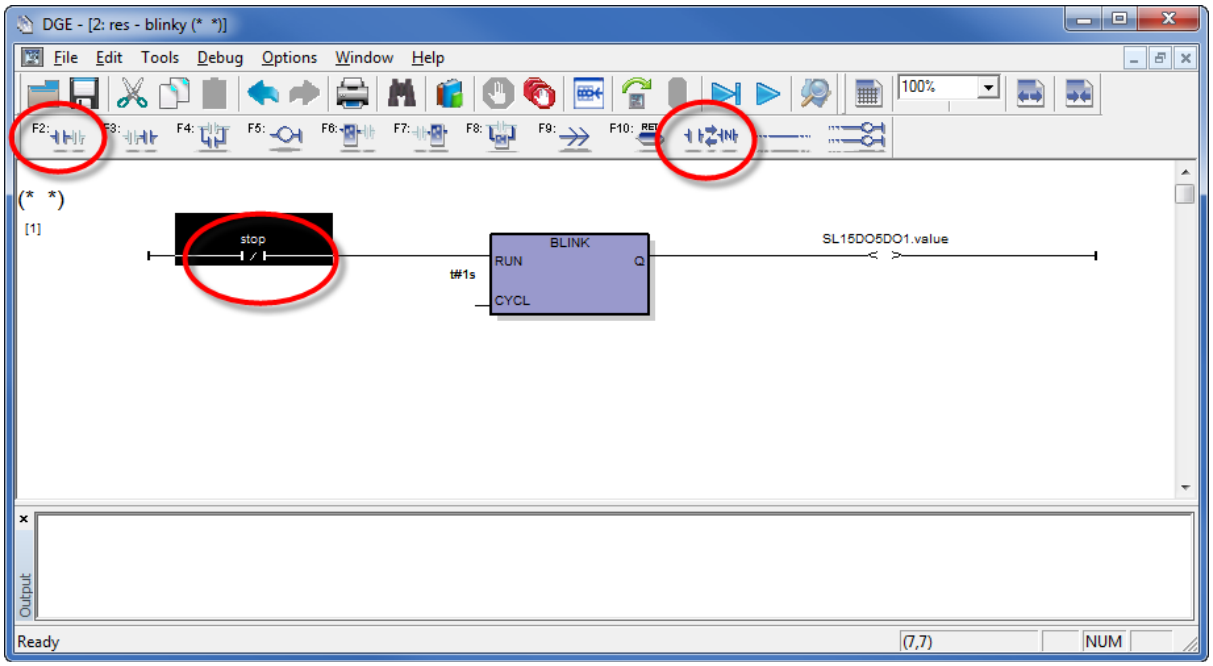

Because the *RUN* input to the *BLINK* function block is connected directly to the left hand "power rail", the function block will be permanently active, so the LED will start flashing as

soon as the logic is loaded onto the RTU. Just for fun, we will now add a contact in series with the function block which will allow us to stop the flashing during operation.

Start by clicking once on the *BLINK* function block to select it, then clicking the leftmost toolbar button, which will insert a contact on the left of the selected object.

In this case we will then change the contact to an “inverted” or “normally closed” contact by clicking the toolbar button once so that a diagonal line symbol is displayed inside the contact.

Finally, we can create a new variable to control the contact by double clicking on the contact and then entering a name, say *stop* into the *Select variable* dialog. By default, the newly created variable will be of type BOOL (Boolean), and will have global scope, meaning it can be used in any program within the RTU.




Because the contact is inverted, when we set the *stop* variable to TRUE the contact will open and the function block will be disabled.

Logic entry is now complete so you can close all ISaGRAF windows and return to Toolbox PLUS+. You will be prompted to save the ISaGRAF project.

### 7.3.3 Downloading to the RTU

The process of downloading a new configuration to the RTU is largely automatic. Simply click the *Download* toolbar button then select *Configuration and Logic.* You may be prompted to first rebuild the ISaGRAF project. Once the required files have been built they will be transferred to the RTU, normally via an Ethernet network connection. The RTU will then update its configuration then automatically restart.

Following the restart, you should see LED 1 on the digital output module start flashing, indicating that the logic has been successfully loaded.

Note that in order to download a new configuration to the RTU you will need to know the current IP address of the CP30. If this is not the same as the address you have set in the

port configuration then you need to tell Toolbox PLUS+ to use the existing address by clicking the *Connection* toolbar button and entering the existing address.

If you don't know the existing IP address then you have a couple of options:

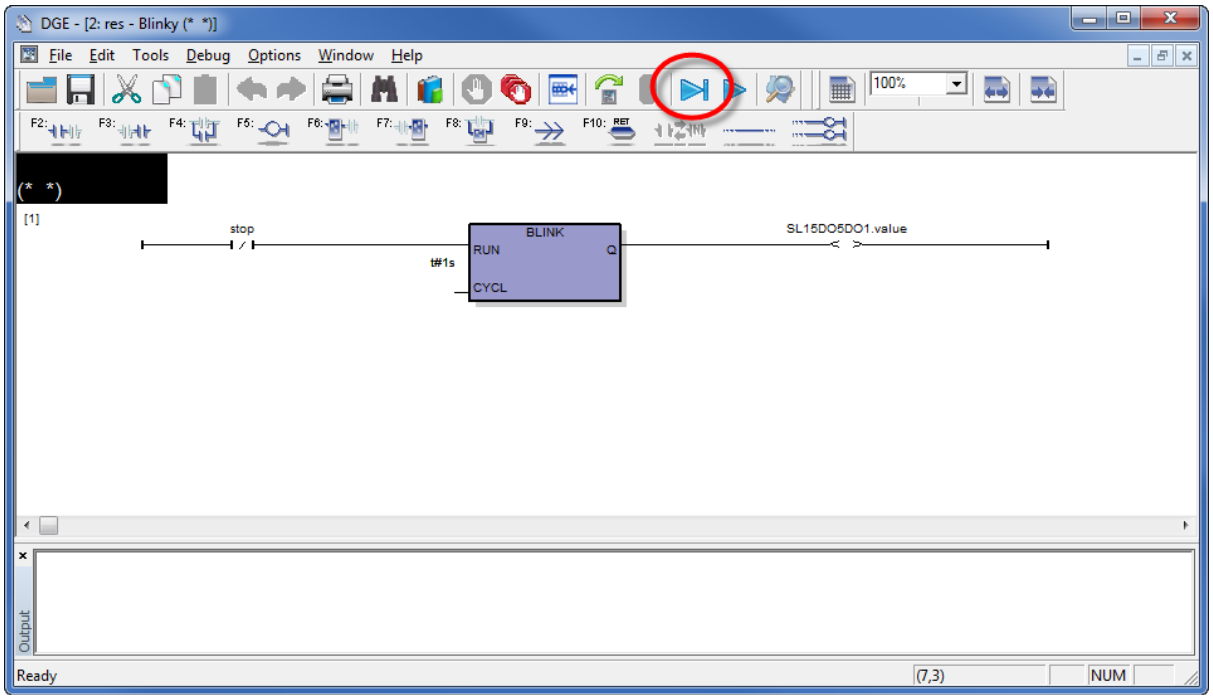
- If the CP30 is connected to the local Ethernet network then you can click *Discovery*, which will attempt to automatically detect the CP30.
- You can reset the CP30 to the factory default address of 192.168.0.1 by removing the battery link located next to the rear connector, waiting a few minutes then replacing it.

### 7.3.4 Debugging with ISaGRAF

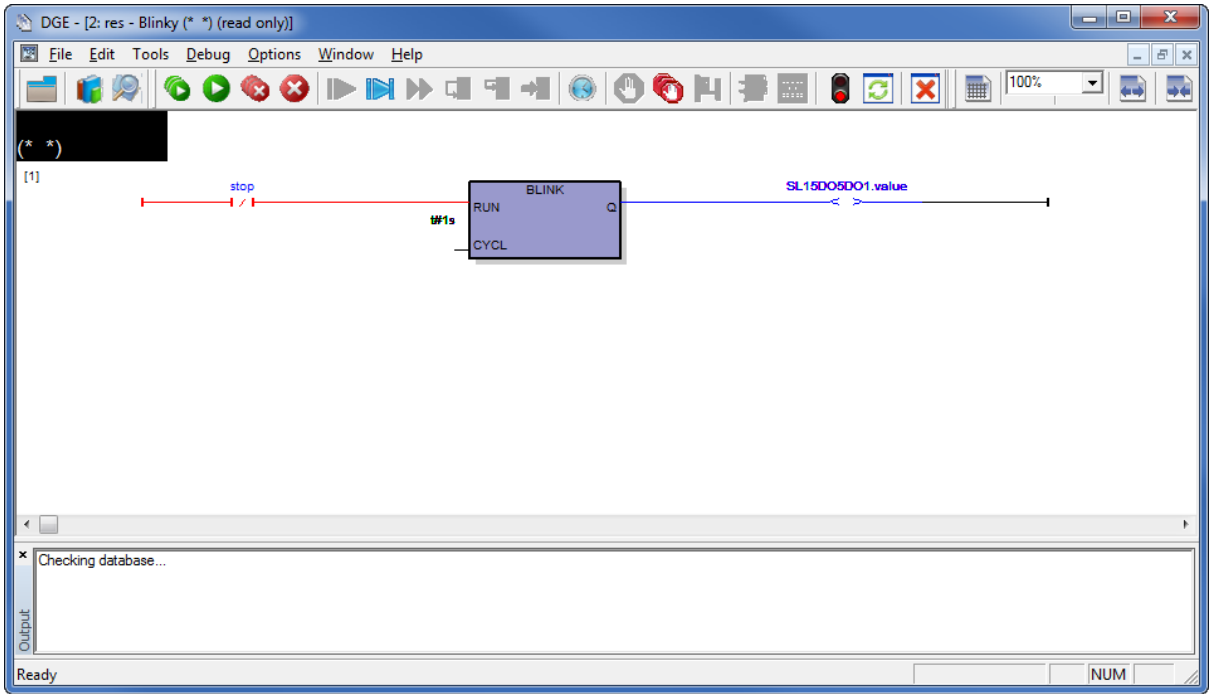
Once a logic program has been downloaded to the RTU, ISaGRAF Workbench can then connect to the RTU, using the same Ethernet connection that was used to download the program. This allows you to view and modify the state of variables in real time, which can be very useful for checking that the logic is operating as expected.

After the logic has been downloaded and has started running, click the *ISaGRAF* button to start ISaGRAF Workbench.

Double click on the name of your program to open the Ladder editor. Then click the *Debug Target* button, as shown below.

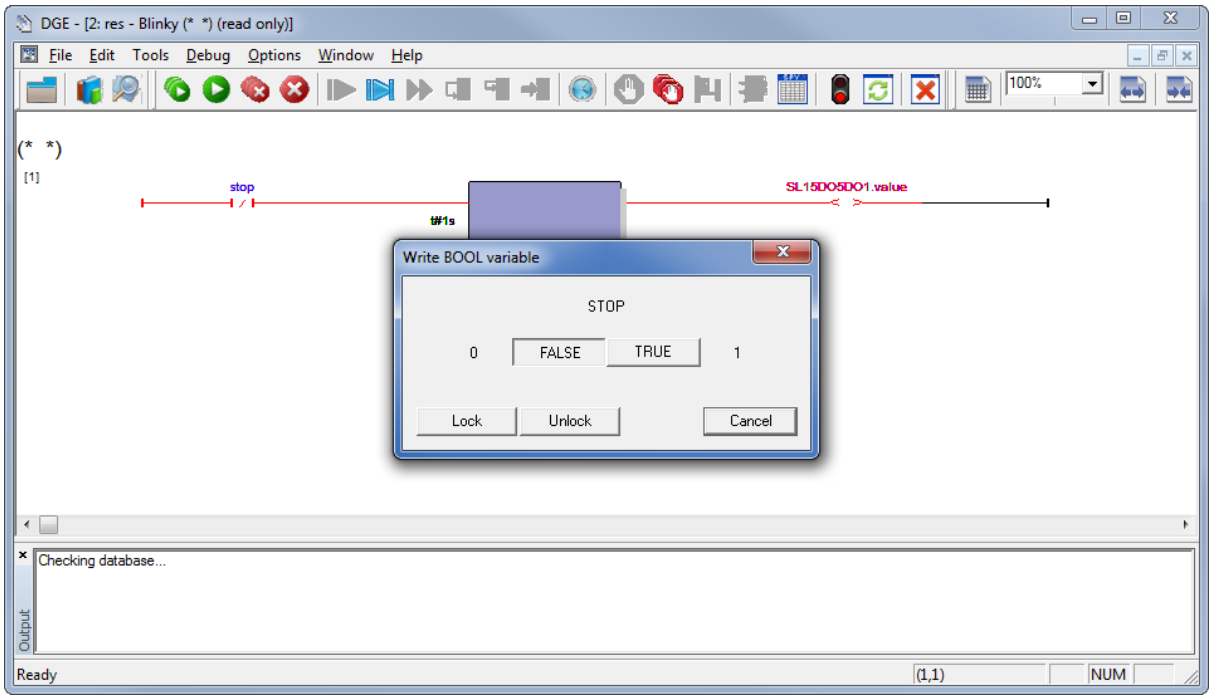



After a short pause the logic will be re-displayed, but this time the rungs will be colour coded according to their current state – red for on (true), blue for off (false).




In this case the input to the function block is red (on), and the output will be flashing red and blue, mirroring the flashing of the LED on the digital output module. (Actually there will always be a small amount of “lag” due to the time taken to communicate the states over the Ethernet network, so for fast changing logic some transitions may not be visible in the ISaGRAF window.)

To modify a value, double click the *stop* contact, which will cause a status window to pop up:




This shows that the current state of the *stop* variable is FALSE, which, because the contact is inverted, causes the input to the function block to be true. If you now click *TRUE*, the variable’s state will be changed and you should observe that the LED on the digital output module stops flashing.

The *Lock* button allows you to force a variable to a particular state, overriding anything that might be setting it in the logic program. An asterisk is displayed next to any variable that is currently locked.

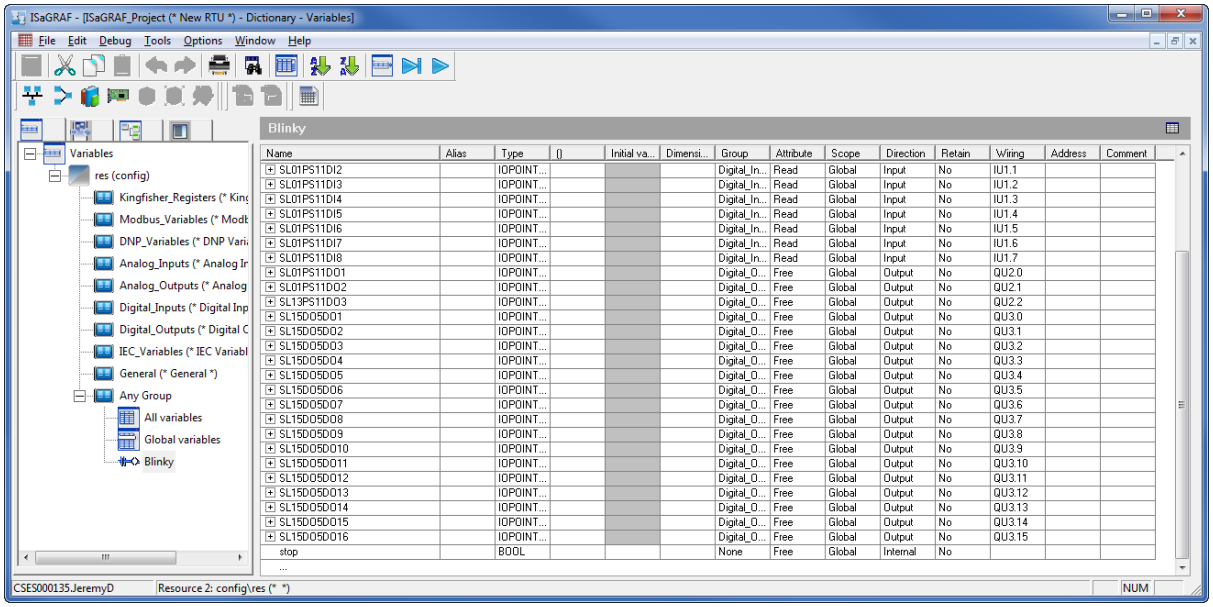
To end a debugging session, click (*Stop Debug Target*), or close the ISaGRAF windows.

Note: In order to successfully debug in ISaGRAF, the logic shown in ISaGRAF Workbench must exactly match that which is running on the RTU. If it doesn't then ISaGRAF will display a warning message. In particular, if you change any logic or variables in ISaGRAF then in order to debug it you will need to first exit ISaGRAF, then download the logic to the RTU using Toolbox PLUS+.

### 7.3.5 ISaGRAF Dictionary

To display a list of all defined variables, click (*Dictionary*) to display the ISaGRAF dictionary.

Note: If you click the Dictionary toolbar button an editor window (e.g. Ladder Editor), ISaGRAF may not automatically switch to the main window containing the Dictionary display. Click the button on the Windows task bar to switch to the main window.


ISaGRAF - [ISaGRAF_Project (* New RTU *) - Dictionary - Variables]
File Edit Debug Tools Options Window Help
Blinky
Variables
res (config)
Kingfisher_Registers (* Kin
Modbus_Variables (* Modb
DNP_Variables (* DNP Vari
Analog_Inputs (* Analog Ir
Analog_Outputs (* Analog
Digital_Inputs (* Digital Inp
Digital_Outputs (* Digital C
IEC_Variables (* IEC Variabl
General (* General *)
Any Group
All variables
Global variables
Blinky
Name Alias Type [] Initial va... Dimensi... Group Attribute Scope Direction Retain Wiring Address Comment
SL01PS11DI2 IOPOINT... Digital_In... Read Global Input No IU1.1
SL01PS11DI3 IOPOINT... Digital_In... Read Global Input No IU1.2
SL01PS11DI4 IOPOINT... Digital_In... Read Global Input No IU1.3
SL01PS11DI5 IOPOINT... Digital_In... Read Global Input No IU1.4
SL01PS11DI6 IOPOINT... Digital_In... Read Global Input No IU1.5
SL01PS11DI7 IOPOINT... Digital_In... Read Global Input No IU1.6
SL01PS11DI8 IOPOINT... Digital_In... Read Global Input No IU1.7
SL01PS11DO1 IOPOINT... Digital_O... Free Global Output No QU2.0
SL01PS11DO2 IOPOINT... Digital_O... Free Global Output No QU2.1
SL13PS11DO3 IOPOINT... Digital_O... Free Global Output No QU2.2
SL15D05DO1 IOPOINT... Digital_O... Free Global Output No QU3.0
SL15D05DO2 IOPOINT... Digital_O... Free Global Output No QU3.1
SL15D05DO3 IOPOINT... Digital_O... Free Global Output No QU3.2
SL15D05DO4 IOPOINT... Digital_O... Free Global Output No QU3.3
SL15D05DO5 IOPOINT... Digital_O... Free Global Output No QU3.4
SL15D05DO6 IOPOINT... Digital_O... Free Global Output No QU3.5
SL15D05DO7 IOPOINT... Digital_O... Free Global Output No QU3.6
SL15D05DO8 IOPOINT... Digital_O... Free Global Output No QU3.7
SL15D05DO9 IOPOINT... Digital_O... Free Global Output No QU3.8
SL15D05DO10 IOPOINT... Digital_O... Free Global Output No QU3.9
SL15D05DO11 IOPOINT... Digital_O... Free Global Output No QU3.10
SL15D05DO12 IOPOINT... Digital_O... Free Global Output No QU3.11
SL15D05DO13 IOPOINT... Digital_O... Free Global Output No QU3.12
SL15D05DO14 IOPOINT... Digital_O... Free Global Output No QU3.13
SL15D05DO15 IOPOINT... Digital_O... Free Global Output No QU3.14
SL15D05DO16 IOPOINT... Digital_O... Free Global Output No QU3.15
stop BOOL None Free Global Internal No
CSES000135.JeremyD
Resource 2: config\res (* *)
NUM

To filter the display so that only a certain group of variables (e.g. Modbus variables) are displayed, click the group name in the left hand pane.

This window may also be used during a debugging session, in which case the current variable values will be displayed.

## 7.4 How Logic is Executed

The RTU repeatedly evaluates the configured logic program(s) according to a regular cycle:

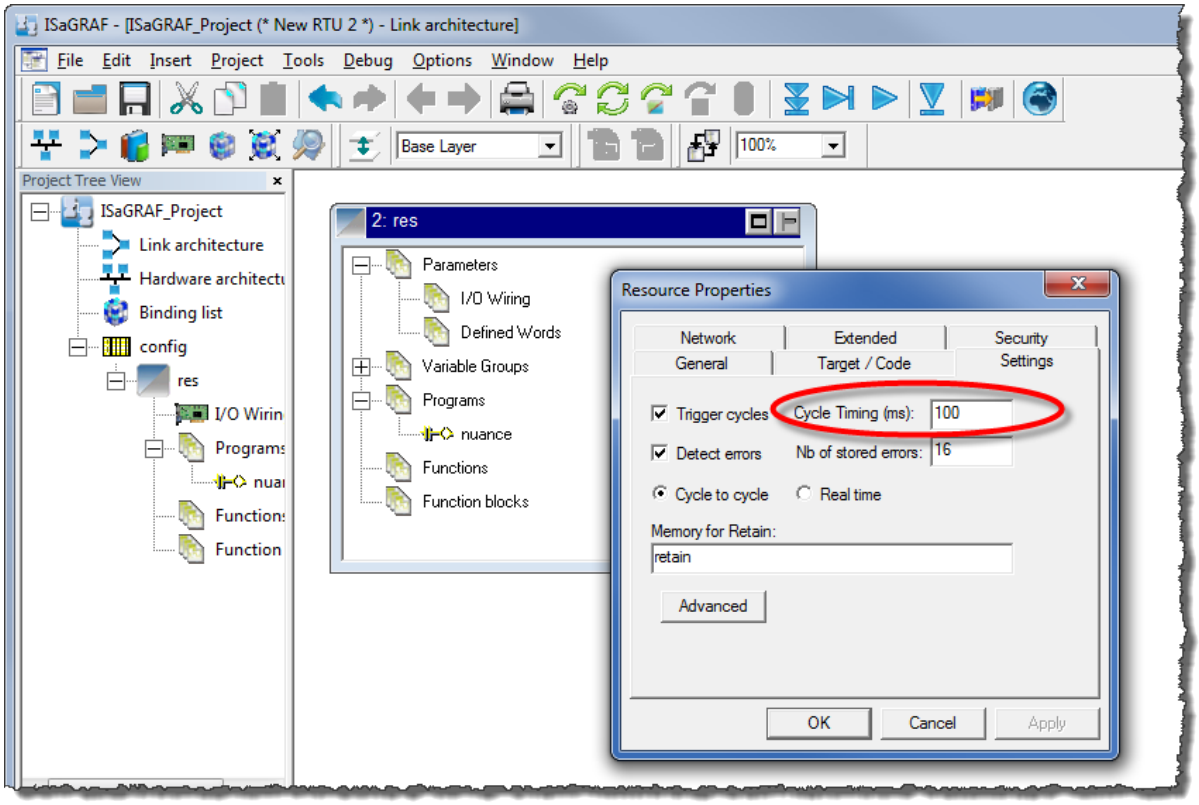
- Read data from I/O modules
- Evaluate logic

- Process received communications messages
- Write data to I/O modules and send outgoing communications messages
- Save retained variables (Using the ISaGRAF *Dictionary* window, variables may be designated as “retained”. Such variables will then keep their value even if the RTU is restarted.)

The above cycle is normally triggered at a fixed rate. By default, the RTU will attempt to execute a new cycle every 100ms.

Note however that in projects with a substantial amount of logic defined, the above tasks may take longer than 100ms – in which case the logic cycles will be executed back to back, as quickly as possible.

To make heavily loaded systems more deterministic, the cycle trigger period can optionally be increased in ISaGRAF Workbench. On the main ISaGRAF screen, select the *Resource* window (normally labelled “res”) and right click the title bar. Select *Properties* to bring up the *Resource Properties* window, then click the *Settings* tab.




For example, if you know that your logic takes about 250ms to execute then you could set the *Cycle Timing* control to 300ms.

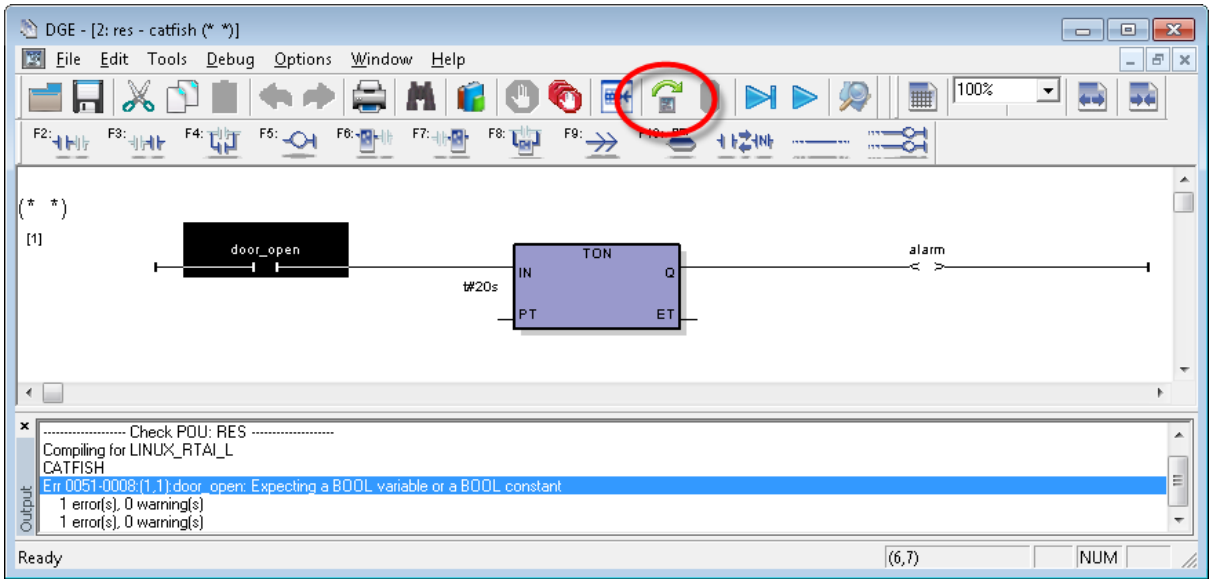
During an ISaGRAF debugging session, you can determine the current approximate cycle time by selecting the *Resource* window, then clicking the *Debug* menu and selecting *Diagnosis*. The *Timing* tab will show the current and maximum cycle execution time. These figures should, however, only be used as a rough guide.

## 7.5 ISaGRAF Tips

For more information about using ISaGRAF, explore the online help within ISaGRAF workbench.

The following general tips may also be useful:

- In Ladder Diagram and Structured Text programs, descriptive comments can (and should) be inserted between (* and *) delimiters. In Function Block Diagrams, comment blocks can be used to insert comments.
- Function block parameters are configured by double-clicking outside the function block, next to the parameter.
- The parameter's data type must match that expected by the function block. Check the documentation of the function block carefully – for RTU specific functions see ISaGRAF Function Blocks in this manual; for standard ISaGRAF functions refer to the ISaGRAF online help.
- A function block parameter may be either:
  - a constant (e.g. *24*, *'5:1'*, *1.0* and *t#5s* are sample integer, string, real and time constants, respectively)
  - a variable (e.g. *SL05DO5DO9.value*, *pump_on*, *rxdata.4* – the latter format specifies bit 4 within the integer variable *rxdata*)
  - a defined word (e.g. *PROTOCOL_DNP3*) These are simply named constants – see ISaGRAF Defined Words.
- New variables can be added to the dictionary in several different ways:
  - implicitly, when a module is added in Toolbox PLUS+
  - in Toolbox PLUS+, by clicking *New* on the *Dictionary* page
  - from within an ISaGRAF logic editor, e.g. by double clicking on a contact in a ladder diagram
  - from within the ISaGRAF dictionary window, by double clicking on the "…" at the bottom of the table
- If you need to create or modify a large number of variables then it may be easier to export the dictionary to an Excel spreadsheet file, edit it in Excel, then re-import it. This is done in Toolbox PLUS+, see File Menu.
- The (*Search*) toolbar button can be used to locate where variables are used, or globally rename all references to a variable.
- A program can be checked while ISaGRAF is running by clicking (*Build*). ISaGRAF will report 0 errors and 0 warnings when the program is OK. If there are errors (as shown below), double-click on the error in the Output window and ISaGRAF will highlight where the error has occurred.




- Toolbox PLUS+ is locked while ISaGRAF is running, and most toolbar buttons are disabled. To continue using Toolbox PLUS, ISaGRAF must first be shut down.
- See ISaGRAF - Logic Examples for sample logic to implement a number of common applications.

## 7.6 ISaGRAF Variable Types

### 7.6.1 Standard variable types

The following IEC 61131 variable types are supported by ISaGRAF.

| Type | Description | Data Range |
|---|---|---|
| BOOL | Logic (true or false) | 1 = True, 0 = False |
| SINT | Signed 8-bit integer | -128 to +127 |
| USINT, BYTE | Unsigned 8-bit integer | 0 to 255 |
| INT | Signed 16-bit integer | -32768 to 32767 |
| UINT, WORD | Unsigned 16-bit integer | 0 to 65535 |
| DINT | Signed 32-bit integer | -2,147,483,648 to +2,147,483,647 |
| UDINT, DWORD | Unsigned 32-bit integer | 0 to 4,294,967,295 |
| LINT (*) | Signed 64-bit integer | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 |
| ULINT, LWORD (*) | Unsigned 64-bit integer | 0 to 18,446,744,073,709,551,615 |
| REAL | 32-bit floating point value (7 significant digits) | 3.4E-37 to 3.4E+37 |
| LREAL (*) | 64-bit floating point value (15 significant digits) | 1.7E-308 to 1.7E+308 |
| TIME | Unsigned 32-bit time value (milliseconds) | 0 to 4,294,967,294 ms (1193h2m47s294ms) |
| DATE | Unsigned 32-bit date/time value (seconds since 0:00UTC 1-Jan-1970) | 0 to 4,294,967,294 s (1-Jan-1970 to 19-Jan-2038) |
| STRING (*) | Character string with a defined maximum size. When defining a string, the maximum size is specified in parentheses e.g. MyString(12) | Up to 255 characters |

(*) Note: 64-bit and string variables may be used in ISaGRAF logic, but event logging and returning of these types via protocols such as DNP3 are not currently supported.

### 7.6.2 Other variable types

The following array and structure types are also supported

| Type | Definition | Description |
|---|---|---|
| TIMESTRUC | DINT seconds | High resolution date/time value, consisting of seconds since 0:00UTC 1-Jan-1970, plus a millisecond value (0-999) |
| | INT millisec | |
| IOPOINT_B | BOOL value | Structure type for holding a BOOL I/O point value, plus additional details such as timestamp and flags. See also IOPOINT types. |
| | TIMESTRUC timestamp | |
| | USINT flags | |
| | USINT datatype | |
| IOPOINT_D | DINT value | Structure type for holding a DINT I/O point value, plus additional details such as timestamp and flags. See also IOPOINT types. |
| | TIMESTRUC timestamp | |
| | USINT flags | |
| | USINT datatype | |
| IOPOINT_R | REAL value | Structure type for holding a REAL I/O point value, plus additional details such as timestamp and flags. See also IOPOINT types. |
| | TIMESTRUC timestamp | |
| | USINT flags | |
| | USINT datatype | |
| INT_ARRAY | Array of 32 INT variables | Used for specifying Kingfisher register numbers or RTU addresses when using KF_RX_DATA, KF_TX_DATA, KF_NW_RX_DATA and KF_NW_TX_DATA function blocks. |
| EVENT_FILTER | DINT value | Used for specifying a filter when retrieving event logs using KF_RX_UPDATE_SINGLE and KF_RX_EVENT_LOGS function blocks:<br>If *all* is set then all events are retrieved. Otherwise, if *greater_equal_than* is set then select events where the parameter >= *value*. Otherwise, if *less_equal_than* is set then select events where the parameter <= *value*. Otherwise, if *equal* is set then select events where the parameter = *value*. |
| | BOOL all | |
| | BOOL greater_equal_than | |
| | BOOL less_equal_than | |
| | BOOL equal | |
| DATA_TYPE | USINT rtunet_usint | Used for specifying a data value when using KF_SET_VARIABLE function block. Only one of these fields should be set.<br>Note: *rtunet_lreal* is not implemented |
| | SINT rtunet_sint | |
| | UINT rtunet_uint | |
| | INT rtunet_int | |
| | UDINT rtunet_udint | |
| | DINT rtunet_dint | |
| | REAL rtunet_real | |
| | LREAL rtunet_lreal | |
| | STRING(255) rtunet_string | |
| | DATE rtunet_date | |
| | TIME rtunet_time | |

### 7.6.3 IOPOINT types

Data values originating from an I/O module or DNP3 are represented using IOPOINT_*x* types (IOPOINT_B for BOOL variables, IOPOINT_D for DINT and IOPOINT_R for REAL). These structures contain the following fields:

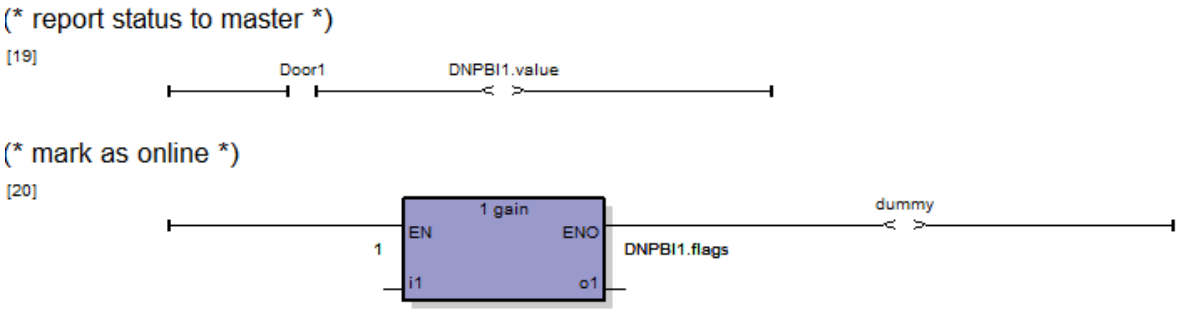
- *value* – the current value
- *timestamp* – the time at which the value was acquired (read from an I/O module or extracted from a received DNP3 message)
- *flags* – a set of 8 DNP3-format status bits:

| Bit | Name | Description |
|---|---|---|
| 0 | ONLINE | a valid value has been read from I/O module or DNP3 message |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | OVER_RANGE | value is over-range (analog I/O module points only) |
| 6 | | |
| 7 | STATE | copy of value (digital points only) |

- *datatype* – for I/O module points only, indicates the category and data type:

| Bit | Name | Description |
|---|---|---|
| 0 | category | 0=analog input, 1=analog output, 2=digital input, 3=digital output, 4=counter input |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | type | 0=signed integer, 2=unsigned integer, 3=floating point |
| 5 | | |
| 6 | | |
| 7 | | |

Note: If you set the *value* field of a DNP3 variable in logic, then by default all other fields will be zero. This means that when the master system polls the variable, it will generally mark the value as “bad” or “offline”, because the ONLINE bit is not set. To rectify this, set the *flags* field in logic as well. For example:



## 7.7 ISaGRAF Constants

Constant values should be specified as follows:

| Type | Sample Constants | Description |
|---|---|---|
| BOOL | TRUE, FALSE | Boolean value |
| Integer types | 0, -5, 20000 | Decimal value |
| | 16#2CFF | Hexadecimal value |
| | 2#0011_1110 | Binary value |
| | DNP_CLASS_1, ROUTE_DIRECT | Defined words representing particular integer values (refer to function block definitions) |
| Floating point (Real) types | 0.0, 3.1415, -2.0 | Decimal format, must include decimal point |
| | 1.0E-3, 9.222e6, -2.1e+8 | Exponential format, must include decimal point |
| TIME | T#0s, T#1m30s, T#90m, T#800ms | Time unit symbols are: d, h, m, s, ms |
| DATE | D#2013-3-28 | Year-month-day |
| STRING | 'cats and dogs'<br>'dollar$$ and newline$N',<br>'beep$07' | Enclose strings in single quotes<br>Prefix special characters with $:<br>$$ – $ character<br>$N – newline (CR,LF)<br>$T – tab character<br>$' – apostrophe character<br>$nn – ASCII character nn (hexadecimal) |

The following Defined Words (symbolic constants) can be used instead of numeric constants.

| Defined Word | Value | Used for Function Blocks |
|---|---|---|
| PROTOCOL_KINGFISHER | 1 | KF_GET_ROUTE, KF_SET_ROUTE, KF_GET_PENDING, KF_CLEAR_PENDING |
| PROTOCOL_DNP3 | 8 | |
| PROTOCOL_SNMPC | 12 | |
| PROTOCOL_MODBUS_ASCII | 14 | |
| PROTOCOL_MODBUS_RTU | 15 | |
| PROTOCOL_MODBUS_TCP | 16 | |
| PROTOCOL_AB | 18 | |
| PROTOCOL_HART | 19 | |
| DNP_CLASS_0 | 1 | DNPM_CLASS_POLL, DNPM_UNSOL_ENABLE, DNPM_UNSOL_DISABLE, DNPS_UNSOL_ENABLE, DNPS_UNSOL_DISABLE |
| DNP_CLASS_1 | 2 | |
| DNP_CLASS_2 | 4 | |
| DNP_CLASS_3 | 8 | |
| DNP_CTRL_PULSE_ON | 1 | DNPM_BINARY_COMMAND |
| DNP_CTRL_PULSE_OFF | 2 | |
| DNP_CTRL_LATCH_ON | 3 | |
| DNP_CTRL_LATCH_OFF | 4 | |
| DNP_AUTO_NONE | 0 | DNPM_BINARY_COMMAND, DNPM_ANALOG_16BIT_COMMAND, DNPM_ANALOG_32BIT_COMMAND, DNPM_ANALOG_FLOAT_COMMAND |
| DNP_AUTO_OPERATE | 1 | |
| DNP_AUTO_FEEDBACK | 2 | |
| DNP_FC_SELECT | 3 | DNPM_BINARY_COMMAND, DNPM_ANALOG_16BIT_COMMAND, DNPM_ANALOG_32BIT_COMMAND, DNPM_ANALOG_FLOAT_COMMAND |
| DNP_FC_OPERATE | 4 | |
| DNP_FC_DIRECT | 5 | |
| ROUTE_DIRECT | 0 | KF_GET_ROUTE, KF_SET_ROUTE |
| ROUTE_INDIRECT | 1 | |

## 7.8 ISAGRAF Reserved Names

Listed below are the names that are reserved by ISaGRAF 5.xx and so cannot be used as variable names. In addition, all names beginning with the underscore character are reserved.

| | |
|---|---|
| **A** | ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN |
| **B** | BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE |
| **C** | CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS |
| **D** | DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD |
| **E** | ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT |
| **F** | FALSE, FIND, FOR, FUNCTION |
| **G** | GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT |
| **I** | IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME |
| **J** | JMP, JMPC, JMPCN, JMPN, JMPNC |
| **L** | LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD |
| **M** | MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX |
| **N** | NE, NOT |
| **O** | OF, ON, OR, OR_MASK, ORN |
| **P** | R, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR |
| **S** | S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SUB_DATE_DATE, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM |
| **T** | TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TYPE |
| **U** | UDINT, UINT, ULINT, UNTIL, USINT |
| **V** | VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, VAR_OUTPUT |
| **W** | WHILE, WITH, WORD |
| **X** | XOR, XOR_MASK, XORN |

### 7.8.1 ISaGRAF Licensing Details

To use ISaGRAF workbench beyond the 30 day trial period, a Sentinel USB licence dongle is required (pictured left). This should be plugged into a USB port on your computer before starting ISaGRAF.

The licence dongles differ in cost and functionality based on the number of usable I/O points. Options include 64, 256, 1024 or unlimited I/O points, or a restricted personal maintenance dongle that cannot compile programs.

When purchasing a licence dongle, the size of the RTU to be implemented needs to be considered.

The I/O point limit effectively constrains the number of readable and settable registers available to Toolbox PLUS+ and ISaGRAF. Note that unused variables can be deleted from the dictionary to save on licence I/O slots.

For example, a system comprising of a PS-1x, CP-30 and an IO-3 uses 31 I/O points, as follows:

| Module | IO Type | IO Function | No. Of Variables |
|---|---|---|---|
| PS-1x | Analog Inputs 1 to 7 | Current and Voltage measurement | 7 |
| | Digital Inputs 1 to 8 | Charge States and Flags | 8 |
| | Digital Outputs 1 to 3 | Power Control | 3 |
| IO-3 | Analog Inputs 1 to 4 | Misc Analog Input | 4 |
| | Analog Output 1 | Misc Analog Output | 1 |
| | Digital Inputs 1 to 4 | Misc Digital Inputs | 4 |
| | Digital Outputs 1 to 4 | Misc Digital Output | 4 |
| | | **Total:** | **31** |

If this system was not required to monitor any power usage or facilitate a backup battery, the variables assigned to the PS-1x can be removed, freeing up 18 IO points.

For additional information on specific variables on each module, see RTU Variables.

# 8. ISaGRAF Function Blocks

A number of protocols and special functions have been implemented using custom ISaGRAF function blocks as detailed below.

| Category | Function Block | Description |
|---|---|---|
| Kingfisher protocol | KF_RX_DATA | Read Kingfisher registers from a remote RTU |
| | KF_TX_DATA | Send Kingfisher registers to a remote RTU |
| | KF_NW_RX_DATA | Read Kingfisher network registers from a remote RTU |
| | KF_NW_TX_DATA | Send Kingfisher network registers to a remote RTU |
| | KF_RX_UPDATE_SINGLE | Read data/events from a Series 2 RTU |
| | KF_GET_VARIABLE | Read a variable from a remote RTU |
| | KF_SET_VARIABLE | Update a variable on a remote RTU |
| | KF_RX_EVENT_LOGS | Read events from a remote RTU |
| | KF_TX_EVENT_LOGS | Send events to a remote RTU |
| DNP3 protocol | DNPM_CLASS_POLL | Read data (selected classes) from slave device |
| | DNPM_INTEGRITY_POLL | Read data (all classes) from slave device |
| | DNPM_READ_GROUP | Read data (selected types) from slave device |
| | DNPM_ANALOG_16BIT_COMMAND | Set analog value on slave device |
| | DNPM_ANALOG_32BIT_COMMAND | Set analog value on slave device |
| | DNPM_ANALOG_FLOAT_COMMAND | Set analog value on slave device |
| | DNPM_BINARY_COMMAND | Set binary value on slave device |
| | DNPM_FREEZE_COUNTERS | Record snapshot of counter values on slave device |
| | DNPM_UNSOL_ENABLE | Enable unsolicited reporting by slave device |
| | DNPM_UNSOL_DISABLE | Disable unsolicited reporting by slave device |
| | DNPM_COLD_RESTART | Restart slave device |
| | DNPM_WARM_RESTART | Restart slave device |
| | DNPM_CLEAR_RESTART | Clear Device Restart IIN bit on slave device |
| | DNPM_LINK_RESET | Reset comms link |
| | DNPS_NEED_TIME | Set Need Time IIN bit |
| | DNPS_UNSOL_ENABLE | Enable unsolicited reporting |
| | DNPS_UNSOL_DISABLE | Disable unsolicited reporting |
| Modbus protocol | MODBUS | Read/write data to/from slave Modbus device |
| Allen Bradley DF1 protocol | ABDF1_RX | Read data from Allen Bradley PLC |
| | ABDF1_TX | Write data to Allen Bradley PLC |
| HART protocol | HART | Read/write data to/from slave HART device |
| SNMP Client protocol | SNMP_GET_INT | Read integer from remote device |
| | SNMP_GET_UINT | Read unsigned integer from remote device |
| | SNMP_GET_STRING | Read string from remote device |
| | SNMP_GET_OBJID | Read OID string from remote device |
| | SNMP_SET_INT | Write integer to remote device |
| | SNMP_SET_UINT | Write unsigned integer to remote device |
| | SNMP_SET_STRING | Write string to remote device |
| | SNMP_SET_OBJID | Write OID string to remote device |
| SNMP Trap protocol | SNMP_GET_TRAP | Read received trap message |
| | SNMP_SEND_TRAP | Send trap message |
| SNMP RMS protocol | SNMPRMS | Send RMS Systems trap message |
| User Defined | USER_TX | Send bytes to remote device |

| | | |
|---|---|---|
| protocol | USER_RX | Read bytes received from remote device |
| | USER_RX_BYTES | Return number of received bytes |
| VRRP protocol | VRRP | Return VRRP status |
| General Communications | KF_GET_COMM_STATS | Get comms statistics for one or all RTUs |
| | KF_RESET_COMM_STATS | Clear comms statistics for one or all RTUs |
| | KF_GET_PORT_STATS | Get comms statistics for a physical port |
| | KF_RESET_PORT_STATS | Clear comms statistics for a physical port |
| | KF_GET_PENDING | Indicates whether a message is in progress |
| | KF_CLEAR_PENDING | Clear the message pending flag |
| | KF_GET_ROUTE | Get information about the route to a remote RTU |
| | KF_SET_ROUTE | Modify the route to a remote RTU |
| Event Logging | KF_EVENT_LOG | Store an event log |
| | KF_CLEAR_EVENT_LOGS | Clear all event logs on local or remote RTU |
| | KF_GET_EVENT_LOG_COUNT | Get event log count on local or remote RTU |
| RTU System Data | KF_GET_ADDRESS | Return RTU address |
| | KF_GET_FIRMWARE | Return RTU firmware version |
| | KF_GET_RTU_TYPE | Return RTU type (CP30/G30) |
| | KF_GET_SYSTEMID | Return system ID byte |
| | KF_GET_PROCESSOR | Return processor backplane slot |
| | KF_GET_MODULE_TYPE | Return module type in specified slot |
| | KF_GET_MODULE_OK | Check whether slot contains expected module |
| | KF_RESET_MODULE | Reset/reconfigure module |
| | REBOOT | Reboot processor module |
| | KF_GET_RTC | Get current time as integer |
| | KF_SET_RTC | Set current time as integer |
| | KF_GET_TIME | Get current time as separate components |
| | KF_SET_TIME | Set current time as separate components |
| Maths & Logic | INCREMENT | Increment a value |
| | DECREMENT | Decrement a value |
| | ChangeDetect | Detect change in a value |
| | MulDiv | Scale a floating point value |
| | MULDIV_INT | Scale an integer value |
| | BCD_TO_BINARY | Convert BCD to integer |
| | BINARY_TO_BCD | Convert integer to BCD |
| | FPACK | Join two integers to make floating point value |
| | FUNPACK | Split floating point value into two integers |
| | NBIT | Set/clear single bit |
| | NCBT | Test single bit for 1 |
| | NOBT | Test single bit for 0 |
| PID Controller | KF_PID | Proportional Integral Derivative Controller |
| Gas Flow Calculations | AGA3 | Calculate gas mass flow |
| | AGA5 | Calculate gas heating value |
| | AGA7 | Calculate gas volumetric flow |
| | AGA8_GROSS | Calculate gas compressibility |
| | AGA8_DETAILED | Calculate gas compressibility |
| | AGA11 | Calculate gas volumetric flow (Coriolis meter) |
| | GBT17747_2 | Calculate gas compressibility (Chinese standard) |
| | GBT21446 | Calculate gas flow rate (Chinese standard) |

These function blocks are shown in Ladder Diagram form. However they can be used in any of the languages supported by ISaGRAF.

**Note**: The *EN* (Enable) and *ENO* (Enable Out) parameters on each function block are only present when the block is used in a Ladder Diagram program. The function block will be executed every cycle, while the *EN* input is true. The state of *EN* is copied to the *ENO* output.

See ISaGRAF - Logic Examples for some practical examples of using these function blocks.

Note that there are some function blocks listed in ISaGRAF which are no longer supported and should not be used. See Obsolete Function Blocks for more details.

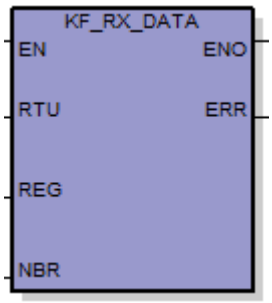
## 8.1 Kingfisher Protocol

The following function blocks are used to initiate Kingfisher protocol operations.

A Kingfisher register variable must be created in the Dictionary for every point that you wish to transfer using Kingfisher protocol.

If the RTU is operating as a slave device then it is not necessary to call these function blocks. Simply load the required values into local Kingfisher registers (KFR*n*), either by mapping them to I/O module points or by setting them in logic.

### 8.1.1 KF_RX_DATA




Polls a remote RTU for up to 32 Kingfisher register values. Registers KFR*n* on the remote RTU#*r* will be transferred to registers KF*r*R*n* on the local RTU.

This is typically used where a master RTU periodically polls an outstation.

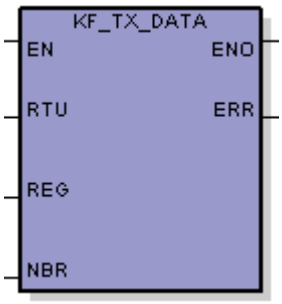
| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU from which to read data (1-65520) |
| **REG** | INT_ARRAY | Array of up to 32 INT values specifying the register numbers (1-2048) to read. |
| **NBR** | USINT | Number of registers to read (1-32) |
| ► **ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

For example, to update registers KF10R1, KF10R2 and KF10R75 (by polling RTU #10), then you would set *RTU*=10, *NBR*=3 and then create an INT_ARRAY variable to specify the three register numbers. This is done from the ISaGRAF Dictionary page. Set the initial values of the array elements as follows:

| Name | Alias | Type | [] | Initial value | Dim |
|---|---|---|---|---|---|
| RTU10regs | | INT_ARRAY | | 1,2,75 | |
| RTU10regs[1] | | INT | | 1 | |
| RTU10regs[2] | | INT | | 2 | |
| RTU10regs[3] | | INT | | 75 | |
| RTU10regs[4] | | INT | | | |
| RTU10regs[5] | | INT | | | |

Then specify the name of the array (*RTU10regs* in this example) for the function block *REG* parameter.

### 8.1.2 KF_TX_DATA



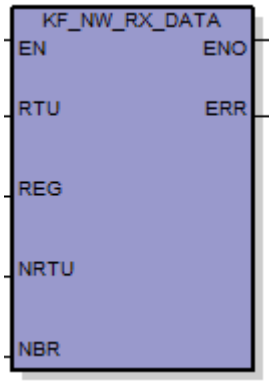

Sends up to 32 Kingfisher local register variables to a remote RTU. Registers KFR*n* on the local RTU will be transferred to registers KF*r*R*n* on the remote RTU (where *r* is the address of the local RTU).

This is typically used for exception reporting, where an outstation reports values to a master when it detects that a change has occurred. It may also be used by a master to update control values on an outstation.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU to send to (1-65520) |
| **REG** | INT_ARRAY | Array of up to 32 INT values specifying the register numbers (1-2048) to send. |
| **NBR** | USINT | Number of registers to send (1-32) |
| ► **ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

For example, to send the values of registers KFR1, KFR2 and KFR75 to an RTU with address 10, you would set *RTU*=10, *NBR*=3 and then create an INT_ARRAY variable to specify the three register numbers, as described in KF_RX_DATA.

### 8.1.3 KF_NW_RX_DATA




Polls a remote RTU for up to 32 Kingfisher register values, which may have originated from other RTUs. Registers KF*r*R*n* on the remote RTU will be transferred to registers KF*r*R*n* on the local RTU.

This is typically used where a master RTU periodically polls a concentrator RTU, which in turn polls various outstation RTUs.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU from which to read data (1-65520) |
| **REG** | INT_ARRAY | Array of up to 32 INT values specifying the register numbers (1-2048) to read. |
| **NRTU** | INT_ARRAY | Array of up to 32 INT values specifying the source RTU (1-249) for each of the registers to read. |
| **NBR** | USINT | Number of registers to read (1-32) |
| ► **ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

For example, suppose RTU #10 is set up to poll RTU #2 and RTU #3. To read and update network registers KF2R1, KF2R2, KF2R42, KF3R1 and KF3R101 (by polling RTU #10) you would set *RTU*=10 and *NBR*=5.

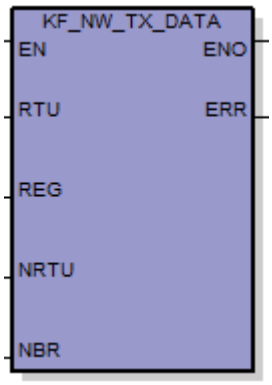
Then create an INT_ARRAY variable to specify the five register numbers, and another INT_ARRAY to specify the source RTU address for each of these five registers. This is done from the ISaGRAF Dictionary page. Set the initial values of the array elements as follows:

| Name | Alias | Type | [] | Initial value | Dim |
|---|---|---|---|---|---|
| RTU10regs | | INT_ARRAY | | 1,2,42,1,101 | |
| RTU10regs[1] | | INT | | 1 | |
| RTU10regs[2] | | INT | | 2 | |
| RTU10regs[3] | | INT | | 42 | |
| RTU10regs[4] | | INT | | 1 | |
| RTU10regs[5] | | INT | | 101 | |
| RTU10regs[6] | | INT | | | |

| Name | Alias | Type | [] | Initial value | Dim |
|---|---|---|---|---|---|
| RTU10addresses | | INT_ARRAY | | 3(2),2(3) | |
| RTU10addresses[1] | | INT | | 2 | |
| RTU10addresses[2] | | INT | | 2 | |
| RTU10addresses[3] | | INT | | 2 | |
| RTU10addresses[4] | | INT | | 3 | |
| RTU10addresses[5] | | INT | | 3 | |
| RTU10addresses[6] | | INT | | | |

Then specify the names of the arrays (*RTU10regs* and *RTU10addresses* in this example) for the function block *REG* and *NRTU* parameters.

### 8.1.4 KF_NW_TX_DATA




Sends up to 32 Kingfisher network register variables to a remote RTU. Registers KF*r*R*n* on the local RTU will be transferred to registers KF*r*R*n* on the remote RTU.

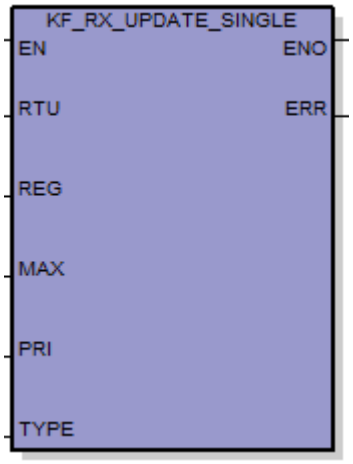
This is typically used for exception reporting, where a concentrator RTU reports values received from one or more outstations to a master RTU.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU to send to (1-65520) |
| **REG** | INT_ARRAY | Array of up to 32 INT values specifying the register numbers (1-2048) to send. |
| **NRTU** | INT_ARRAY | Array of up to 32 INT values specifying the source RTU (1-249) for each of the registers to send. |
| **NBR** | USINT | Number of registers to send (1-32) |
| ► **ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

For example, to send the network registers KF2R1, KF2R2, KF2R42, KF3R1 and KF3R101 to RTU #1 you would set *RTU*=1 and *NBR*=5.

Then create an INT_ARRAY variable to specify the five register numbers, and another INT_ARRAY to specify the source RTU address for each of these five registers, as described in KF_NW_RX_DATA.

### 8.1.5 KF_RX_UPDATE_SINGLE

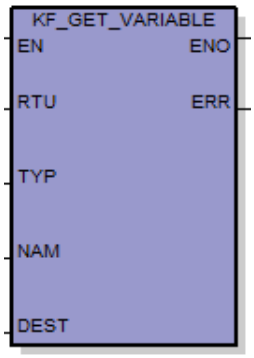



Retrieves data or event logs from a remote RTU.

This block is typically only used when polling a Kingfisher Series 2 RTU (e.g. CP-12, LP-3). For CP-30 based RTUs, DNP3 is normally used for communicating event logs.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU from which to retrieve data (1-65520) |
| **REG** | STRING(16) | String containing the name of an integer variable (e.g. 'ControlReg'). The value of this variable is interpreted as follows::<br>• 0 or 1: The current state of all defined Kingfisher registers on the target RTU will be read, which will then update the corresponding Kingfisher network registers on the local RTU.<br>• 2: Logged events that match the filter settings will be retrieved from the target RTU.<br>• 4: The clock on the target RTU will be synchronised to that on the local RTU. |
| **MAX** | UINT | Maximum number of events to retrieve (if control register = 2) |
| **PRI** | EVENT_FILTER | Retrieve events where the Priority field matches the filter (if control register = 2) |
| **TYPE** | EVENT_FILTER | Retrieve events where the Type field matches the filter (if control register = 2) |
| ► **ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

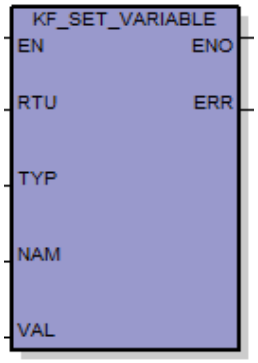
### 8.1.6 KF_GET_VARIABLE




Retrieves a variable by name from a remote CP30/G30 based RTU.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU from which to retrieve variable (1-65520) |
| **TYP** | USINT | Data type of variable:<br>1=USINT, 2=SINT, 3=UINT, 4=INT, 5=UDINT, 6=DINT, 7=REAL, 9=STRING, 10=DATE, 11=TIME. |
| **NAM** | STRING(16) | Name of variable on remote RTU to retrieve. |
| **DEST** | STRING(16) | Name of variable on local RTU to write value to. |
| **► ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

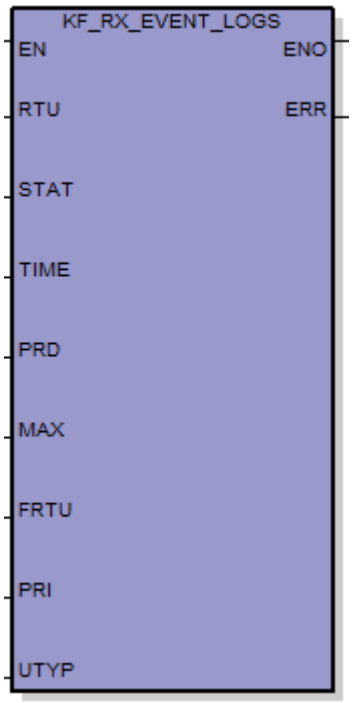
### 8.1.7 KF_SET_VARIABLE




Sets the value of a variable on a remote CP30/G30 based RTU.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU to write to (1-65520) |
| **TYP** | USINT | Data type of variable:<br>1=USINT, 2=SINT, 3=UINT, 4=INT, 5=UDINT, 6=DINT, 7=REAL, 9=STRING, 10=DATE, 11=TIME. |
| **NAM** | STRING(16) | Name of variable on remote RTU to set. |
| **VAL** | DATA_TYPE | Value to set |
| **► ERR** | DINT | Status output (0=OK, -34=one or more parameters out of range) |

### 8.1.8 KF_RX_EVENT_LOGS



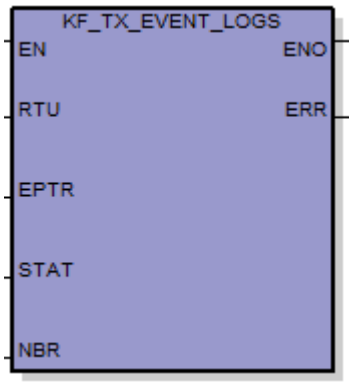

Retrieves event logs that occurred over a specific time period from a remote RTU. It will keep polling event logs until it has received the maximum limit of logs or until the end of the event log list is reached.

This block is typically only used when polling a Kingfisher Series 2 RTU (e.g. CP-12, LP-3). For CP-30 based RTUs, DNP3 is normally used for communicating event logs.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of RTU from which to retrieve logs (1-65520) |
| **STAT** | STRING(16) | String containing the name of an integer variable (e.g. 'StatusReg'). Will be updated, but not with anything useful. |
| **TIME** | UDINT | Start time of the first event log to receive (minutes before now). |
| **PRD** | UDINT | Time period of event logs to receive (minutes). |
| **MAX** | UINT | Maximum number of event logs to retrieve |
| **FRTU** | UINT | If non-zero, only events matching the event filters below will be returned. |
| **PRI** | EVENT_FILTER | Retrieve events where the Priority field matches the filter |
| **UTYP** | EVENT_FILTER | Retrieve events where the Type field matches the filter |
| ► **ERR** | DINT | Status output (0=OK, non-zero if error) |

Note: If the remote RTU address is less than 256 then a "Series 2" event log request is performed. This will only return events relating to Kingfisher registers and I/O module points. If RTU address is greater than or equal to 256 then a "Series 3" request is sent, which will return all event logs.

### 8.1.9 KF_TX_EVENT_LOGS




Sends event logs to a remote RTU.

This block is typically only used when sending data to a Kingfisher Series 2 RTU (e.g. CP-12, LP-3). For CP-30 based RTUs, DNP3 is normally used for communicating event logs.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of destination RTU (1-65520) |
| **EPTR** | STRING(16) | String containing the name of an integer variable (e.g. 'EventPtr'). This variable contains the index of the first event log to send, and will be automatically updated on completion of the function block. |
| **STAT** | STRING(16) | String containing the name of an integer variable (e.g. 'StatusReg'). Set to 128 if an error occurs, set to 1 when transmission is complete. |
| **NBR** | UINT | Maximum number of event logs to send. |
| **► ERR** | DINT | Status output (0=OK, non-zero if parameter error) |

Note: If the remote RTU address is less than 256 then a "Series 2" event log request is performed. Only events relating to Kingfisher registers and I/O module points will be sent. If RTU address is greater than or equal to 256 then a "Series 3" request is sent, which will send all event logs.

## 8.2 DNP3 Protocol

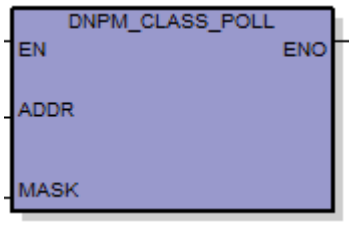
The following function blocks are used to initiate DNP3 protocol operations.

Function blocks with names beginning with DNPM_ are used when the RTU is operating as a DNP3 master, while DNPS_ function blocks are used when the RTU is operating as a DNP3 slave.

A DNP3 variable must be created in the Dictionary for every point that you wish to transfer using the DNP3 protocol.

If the RTU is operating as a slave device then it is not necessary to call these function blocks. Simply load the required values into local DNP3 registers (DNPBI*n*, DNPAI*n* etc.), either by mapping them to I/O module points or by setting them in logic.

### 8.2.1 DNPM_CLASS_POLL




Polls a remote DNP3 device for static values (class 0 data) and/or events (class 1/2/3 data).

Returned object data will be stored in the local RTU if DNP3 variables (e.g. DNP*r*BI*n* for binary inputs) have been created in the Dictionary.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device from which to read data (1-65535) |
| **MASK** | USINT | Bitmask specifying class(es) of data to read (0-15):<br>bit0=Class 0, bit1=Class 1, bit2=Class 2, bit3=Class 3.<br>If a single class is being read then the defined words DNP_CLASS_0, DNP_CLASS_1, DNP_CLASS_2 or DNP_CLASS_3 may also be used. |

For example, to read the current states of all DNP3 variables (i.e. class 0 data) and all class 1 events from a DNP3 device with address 3008 then you would set *ADDR*=3008 and *MASK*=3. If a variable DNP3008BI22 was defined on the local RTU then its value would be updated (assuming that DNP Binary Input #22 was defined on the slave device). If the variable was defined as class 1, then any events received from the device relating to this variable would be logged.

Any received static (class 0) data for which DNP3 variables are not defined will be discarded.

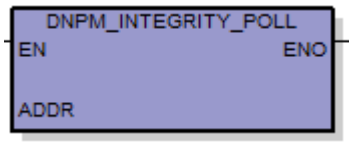
**Note**: The DNP3 protocol does not transmit class information along with each event. When DNP3 events are received and logged by the RTU, the event logs will each be tagged with the class number that was requested (1, 2 or 3). This means that if multiple classes of DNP3 events are requested simultaneously, then the class number that will be attached to the event logs will not necessarily match that of the original point on the remote source RTU.

This can be important where the RTU is operating as a concentrator, i.e polling slave DNP3 devices and in turn being polled by a DNP3 master.

For example, suppose a remote slave device contains two DNP3 points DNPBI101 (class 1) and DNPBI102 (class 2). If the RTU performs a Class 1+2 poll (*MASK*=6) then all events for both these points will be logged as Class 1 events. If a master system then issues a Class 2 poll to the RTU, events for DNPBI102 will not be returned.

If you wish to preserve the original point's class in the event log then separate class 1, 2 and 3 polls should be performed. Likewise, the DNPM_INTEGRITY_POLL function block (which performs a Class 0+1+2+3 poll) should not be used.

### 8.2.2 DNPM_INTEGRITY_POLL

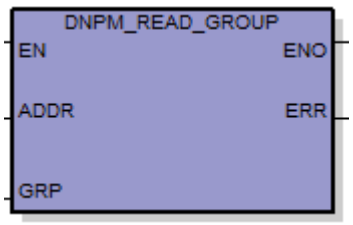



Polls a remote DNP3 device for static values (class 0 data) and events (class 1/2/3 data). Equivalent to calling DNPM_CLASS_POLL with *MASK*=15.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device from which to read data (1-65535) |

**Note**: Events logged by the RTU using this function block will all be marked as Class 1 events, which may not be desirable in situations where the RTU is in turn being polled by a master system. See DNPM_CLASS_POLL for more information.

### 8.2.3 DNPM_READ_GROUP

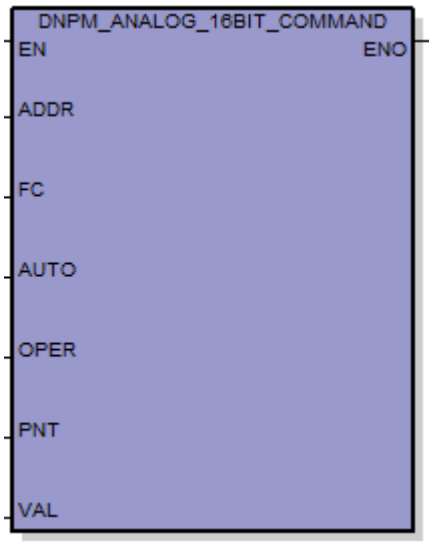



Polls a remote DNP3 device for static values (class 0 data) of a particular type.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device from which to read data (1-65535) |
| **GRP** | USINT | Data type group to poll:<br>1=binary inputs, 10=binary outputs, 20=binary counters, 21=frozen counters, 30=analog inputs, 40=analog outputs |
| ► **ERR** | UINT | Status output (0=OK, 65535=one or more parameters out of range) |

For example, if *ADDR* is set to 205 and *GRP* is set to 30 then analog input variables (DNP205AI*nn*) would be updated.

### 8.2.4 DNPM_ANALOG_16BIT_COMMAND




Sets a 16-bit (UINT) DNP3 variable on a slave device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device containing the variable to set (1-65535) |
| **FC** | USINT | Specifies the function to perform:<br>3=Select, 4=Operate, 5=Direct Operate<br>The defined words DNP_FC_SELECT, DNP_FC_OPERATE and DNP_FC_DIRECT may also be used.<br>See DNP3 Select and Operate for more information. |
| **AUTO** | USINT | Specifies whether the above function should be automatically followed by the appropriate confirmation function.<br>0=None, 1=Automatic Operate after Select, 2=Automatic Feedback Poll after Operate<br>The defined words DNP_AUTO_NONE, DNP_AUTO_OPERATE and DNP_AUTO_FEEDBACK may also be used.<br>See DNP3 Select and Operate for more information. |
| **OPER** | UDINT | If an automatic feedback poll is selected, this parameter specifies the delay (ms) between the Operate and the feedback poll. |
| **PNT** | UINT | DNP3 analog point number (0-65535) |
| **VAL** | UINT | Value to set |

#### *DNP3 Select and Operate*

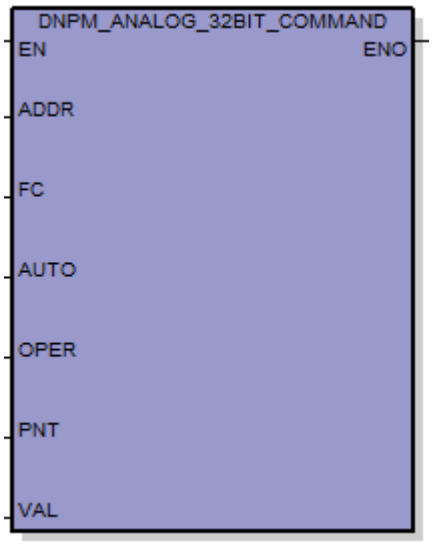
The DNP3 protocol provides an option to set control outputs using a two stage confirmation process. First, a **Select** message is sent which specifies the desired state(s). If a valid acknowledgement is received from the slave then an **Operate** message containing identical data is sent, which the slave will then action.

If this confirmation procedure is not required then a single **Direct Operate** message can be used. The master can then optionally poll the outputs to verify that they were set.

The following combinations of the *FC* and *AUTO* function block parameters may be used:

| *FC* | *AUTO* | Action |
|---|---|---|
| DNP_FC_SELECT | DNP_AUTO_NONE | Send Select message |
| DNP_FC_SELECT | DNP_AUTO_OPERATE | Send Select message. If valid response from slave then send Operate message. |
| DNP_FC_OPERATE | DNP_AUTO_NONE | Send Operate message |
| DNP_FC_OPERATE | DNP_AUTO_FEEDBACK | Send Operate message. Wait for specified delay, then send Feedback Poll message. |
| DNP_FC_DIRECT | DNP_AUTO_NONE | Send Direct Operate message |
| DNP_FC_DIRECT | DNP_AUTO_FEEDBACK | Send Direct Operate message. Wait for specified delay, then send Feedback Poll message. |

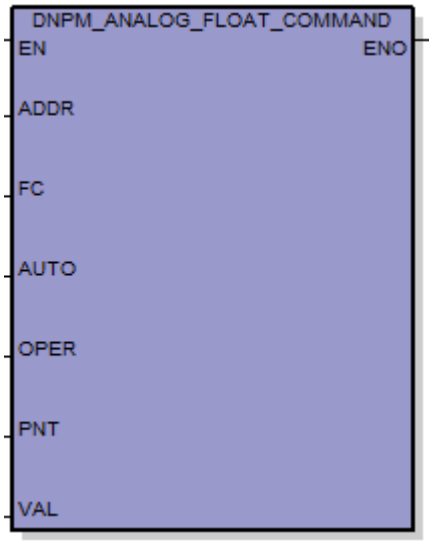
### 8.2.5 DNPM_ANALOG_32BIT_COMMAND




Sets a 32-bit (UDINT) DNP3 variable on a slave device.

Same as DNPM_ANALOG_16BIT_COMMAND, apart from the type of the *VAL* parameter.

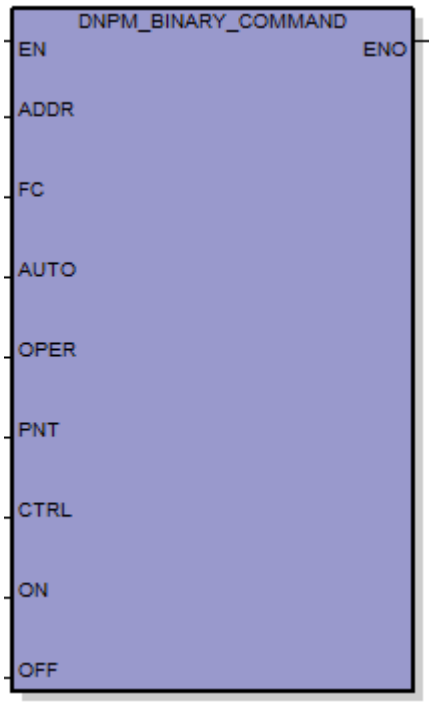
### 8.2.6 DNPM_ANALOG_FLOAT_COMMAND




Sets a 32-bit floating point (REAL) DNP3 variable on a slave device.

Same as DNPM_ANALOG_16BIT_COMMAND, apart from the type of the *VAL* parameter.

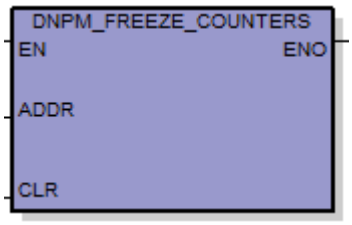
### 8.2.7 DNPM_BINARY_COMMAND




Sets a binary (BOOL) DNP3 variable on a slave device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device containing the variable to set (1-65535) |
| **FC** | USINT | Specifies the function to perform, as per DNPM_ANALOG_16BIT_COMMAND |
| **AUTO** | USINT | Specifies the automatic confirmation function, as per DNPM_ANALOG_16BIT_COMMAND |
| **OPER** | UDINT | Specifies the delay before performing an automatic feedback poll, as per DNPM_ANALOG_16BIT_COMMAND |
| **PNT** | UINT | DNP3 binary point number (0-65535) |
| **CTRL** | USINT | Binary control function to perform:<br>1=Pulse On, 2=Pulse Off, 3=Latch On, 4=Latch Off<br>The defined words DNP_CTRL_PULSE_ON, DNP_CTRL_PULSE_OFF, DNP_CTRL_LATCH_ON and DNP_CTRL_LATCH_OFF may also be used. |
| **ON** | UDINT | If Pulse On is selected, this specifies the pulse on time (ms). |
| **OFF** | UDINT | If Pulse Off is selected, this specifies the pulse off time (ms). |

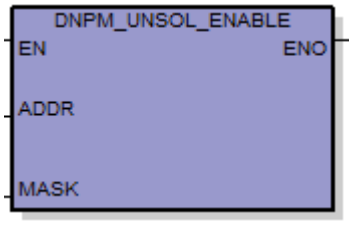
### 8.2.8 DNPM_FREEZE_COUNTERS




Freezes the counters in a slave device, i.e. takes a snapshot of their current count values. Additionally it allows the counters to be cleared after being frozen.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device containing the counter variables (1-65535) |
| **CLR** | BOOL | If true then the counters will be cleared after being frozen. |

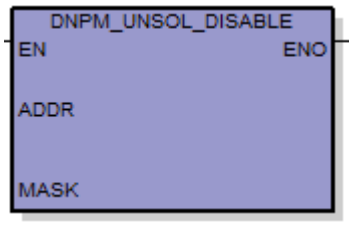
### 8.2.9 DNPM_UNSOL_ENABLE




Commands a slave device to enable unsolicited responses.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |
| **MASK** | USINT | Bitmask specifying class(es) of data for which to enable unsolicited responses (0-15):<br>bit1=Class 1, bit2=Class 2, bit3=Class 3. |

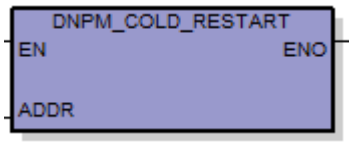
### 8.2.10 DNPM_UNSOL_DISABLE




Commands a slave device to disable unsolicited responses.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |
| **MASK** | USINT | Bitmask specifying class(es) of data for which to disable unsolicited responses (0-15):<br>bit1=Class 1, bit2=Class 2, bit3=Class 3. |

### 8.2.11 DNPM_COLD_RESTART

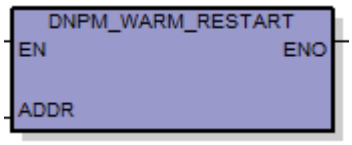



Commands a slave device to perform a cold restart.

Note that a slave CP30 does not distinguish between cold and warm restarts: both requests will cause a reboot.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |

### 8.2.12 DNPM_WARM_RESTART

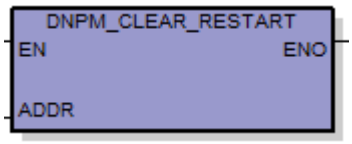



Commands a slave device to perform a warm restart.

Note that a slave CP30 does not distinguish between cold and warm restarts: both requests will cause a reboot.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |

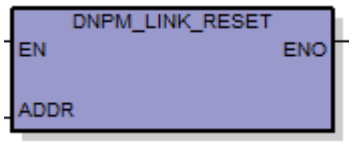
### 8.2.13 DNPM_CLEAR_RESTART




Resets the DEVICE_RESTARTED bit in a slave device's Internal Indications (IIN) register.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |

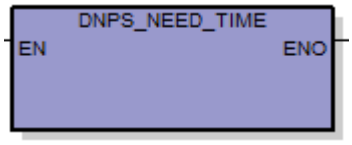
### 8.2.14 DNPM_LINK_RESET




Sends a RESET_LINK_STATES message to a slave device. This will reset the data link layer of the DNP3 protocol.

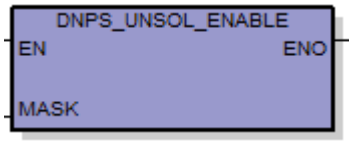
| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of slave device (1-65535) |

### 8.2.15 DNPS_NEED_TIME




Sets the NEED_TIME bit in the local RTU's Internal Indications (IIN) register. When next polled by the master device, the RTU's clock will be synchronised to that of the master.

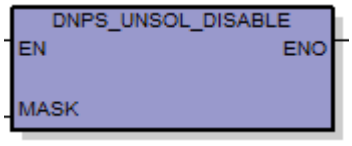
### 8.2.16 DNPS_UNSOL_ENABLE




Enables the sending of unsolicited data by the local RTU.

| Parameter | Type | Description |
|---|---|---|
| **MASK** | USINT | Bitmask specifying class(es) of data for which to enable unsolicited responses (0-15):<br>bit1=Class 1, bit2=Class 2, bit3=Class 3. |

### 8.2.17 DNPS_UNSOL_DISABLE




Disables the sending of unsolicited data by the local RTU.

| Parameter | Type | Description |
|---|---|---|
| **MASK** | USINT | Bitmask specifying class(es) of data for which to disable unsolicited responses (0-15):<br>bit1=Class 1, bit2=Class 2, bit3=Class 3. |

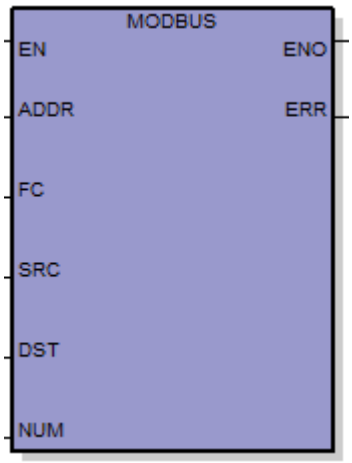
## 8.3 Modbus Protocol

The following function blocks are used to initiate Modbus protocol operations.

A Modbus variable must be created in the Dictionary for every point that you wish to transfer using the Modbus protocol.

If the RTU is operating as a slave device then it is not necessary to call this function block. Simply load the required values into local Modbus registers (MODC*n*, MODI*n* etc.), either by mapping them to I/O module points or by setting them in logic.

### 8.3.1 MODBUS




Sends or receives data in Modbus network registers (MOD*r*C*n*, MOD*r*D*n*, MOD*r*H*n*, MOD*r*I*n*) to/from a Modbus slave device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the Modbus slave device (1-254). If the device is an RTU (address 1-65520) then only the lower 8 bits of the address will be used. |
| **FC** | USINT | Function code. See Supported function codes. |
| **SRC** | UINT | First source Modbus register (1-65535). For read operations this will be a register in the slave device; for writes it will be an RTU Modbus register. |
| **DST** | UINT | First destination Modbus register (1-65535). For read operations this will be an RTU Modbus register; for writes it will be a register in the slave device. |
| **NUM** | USINT | Number of registers to transfer. See Supported function codes. |
| ► **ERR** | DINT | Status code. 0=OK, non-zero indicates a problem with one or more parameters |

Note: In some systems, the type of Modbus point is identified by adding a digit on the front of the register number, as follows:

- coils are numbered 000001-065535 (or 00001-09999)
- discrete inputs are numbered 100001-165535 (or 10001-19999)
- input registers are numbered 300001-365535 (or 30001-39999)
- holding registers are numbered 400001-465535 (or 40001-49999)

The value specified for the *SRC* and *DST* parameters should not include this initial digit, e.g. to specify the first holding register use the value 1, not 40001. (The type of register is implied by the function code.)

### *Supported function codes*

The following function codes are supported:

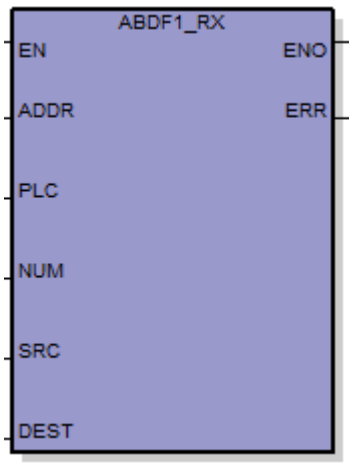
| *FC* | *NUM* | Description |
|---|---|---|
| **1** | 1-128 | Read coils – update RTU variables MOD*r*C*n* (*r*=lower 8 bits of *ADDR, n=DEST* to *DEST+NUM-1*) |
| **2** | 1-128 | Read discrete inputs – update RTU variables MOD*r*D*n* |
| **3** | 1-120 * | Read holding registers – update RTU variables MOD*r*H*n* |
| **4** | 1-120 * | Read input registers – update RTU variables MOD*r*I*n* |
| **5** | 1 | Write coil – copy from RTU variable MOD*r*C*n* (*r*=lower 8 bits of *ADDR, n=SRC)* |
| **6** | 1 | Write holding register – copy from RTU variable MOD*r*H*n* |
| **15** | 1-128 | Write multiple coils – copy from RTU variables MOD*r*C*n* (*r*=lower 8 bits of *ADDR, n=SRC* to *SRC+NUM-1)* |
| **16** | 1-120 * | Write multiple holding registers – copy from RTU variables MOD*r*H*n* |

* Valid range is 1-50 for Modbus/ASCII protocol

## 8.4 Allen Bradley DF1 Protocol

The following function blocks are used to initiate Allen Bradley DF-1 protocol operations.

### 8.4.1 ABDF1_RX



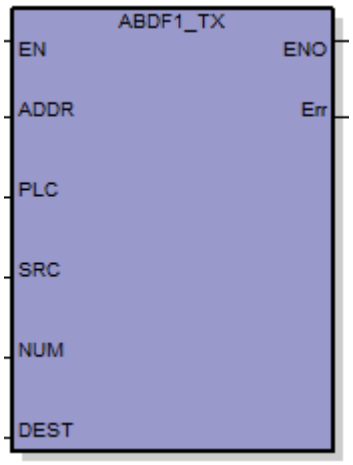

Reads up to 100 consecutive data registers from a remote Allen Bradley PLC using the DF1 protocol.

The returned status code and data values will be stored in local Kingfisher variables (KFR*nn*), which must have been created in the Dictionary.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | DINT | Station address (1-249) configured in the Allen Bradley PLC. An Allen Bradley PLC is treated like another RTU in the network, which means that the station address must be different to all the other addresses in the RTU's Route list. |
| **PLC** | USINT | Not used (set to 0) |
| **NUM** | USINT | Number of registers to read (1-100) |
| **SRC** | STRING(20) | Source address in the Allen Bradley PLC from which to read, e.g. 'N10:1' |
| **DEST** | DINT | Starting Kingfisher register number. A status code (0=OK) is stored in the first register, followed by the *NUM* data values. |
| ► **ERR** | USINT | Not used (always 0) |

For example, if you set *NUM=5* and *DEST*=100 then the status code would be stored in the variable KFR100 and the data values would be stored in KFR101 through KFR105, assuming these variables have been created in the Dictionary.

### 8.4.2 ABDF1_TX



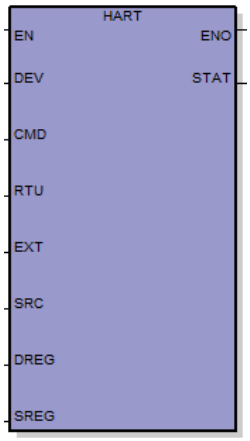

Transmits up to 100 consecutive Kingfisher registers to a remote Allen Bradley PLC using the DF1 protocol.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | DINT | Station address (1-249) configured in the Allen Bradley PLC. |
| **PLC** | USINT | Not used (set to 0) |
| **NUM** | USINT | Number of registers to send (1-100) |
| **SRC** | DINT | Starting Kingfisher register number. |
| **DEST** | STRING(20) | Destination address in the Allen Bradley PLC where data is to be written, e.g. ‘N10:1’ |
| **► Err** | USINT | Not used (always 0) |

## 8.5 HART Protocol

The following function blocks are used to initiate HART protocol operations.

### 8.5.1 HART




Sends or receives data in Kingfisher network registers (KF*r*R*n* or KF*r*F*n*) to/from a slave HART device. This requires the use of a HART communications option board.

| Parameter | Type | Description |
|---|---|---|
| **DEV** | USINT | The HART device address (0-15). Address 0 is only used for point to point installations. |
| **CMD** | USINT | The HART command (or function code) to perform (0-108). See Supported HART Commands. |
| **RTU** | USINT | Data are stored in integer and floating point Kingfisher network registers (KF*r*R*n* and KF*r*F*n*), where *r* is specified by this parameter (1-249). It need not be the same as the actual device address (*DEV* parameter). |
| **EXT** | UINT | Specifies the first of three consecutive Kingfisher registers used to store the HART device's 38-bit extended address, which is retrieved using the Read Unique Identifier command (*CMD*=0).<br>For example, if *RTU*=7 and *EXT*=3 then the extended address would be stored in KF7R3, KF7R4 and KF7R5. |
| **SRC** | UINT | Specifies the first in a block of consecutive Kingfisher registers which hold the data to be sent when performing a Write operation. The number of registers sent and their meanings will vary depending on the selected command.<br>It is recommended that a block of at least 10 integer Kingfisher registers (KF*r*R*n*) be created for HART Write operations. |
| **DREG** | UINT | Specifies the first in a block of consecutive Kingfisher registers which will be updated when performing a Read operation. The number of registers updated and their meanings will vary depending on the selected command.<br>It is recommended that a block of at least 10 integer Kingfisher registers (KF*r*R*n*) and 10 floating point registers (KF*r*F*n*) be created for HART Read operations. |
| **SREG** | UINT | Specifies a Kingfisher register which will be updated with a status code following each command. |
| ► **STAT** | DINT | Not used (always 0) |

For additional information and example projects, please refer to the *HART Implementation Guide,* available from the Kingfisher support website.

### *Supported HART Commands*

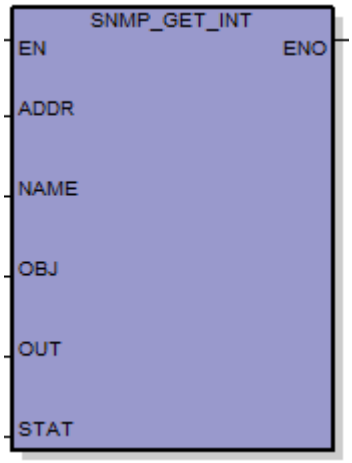
The following function codes are supported:

| *CMD* | Description |
|---|---|
| 0 | Read Unique Identifier |
| 1 | Read PV [primary variable] |
| 2 | Read Current and % of range |
| 3 | Read Current and 4 Dynamic Variables |
| 6 | Write Polling Address |
| 11 | Read Unique Identifier with Tag |
| 12 | Read Message |
| 13 | Read Tag, Descriptor, Date |
| 14 | Read PV Sensor Information |
| 15 | Read [PV] Output Information |
| 16 | Read Final Assembly Number |
| 17 | Write Message |
| 18 | Write Tag, Descriptor, Date |
| 19 | Write Final Assembly Number |
| 33 | Read Transmitter Variables |
| 34 | Write [PV] Damping Value |
| 35 | Write [PV] Range Values |
| 36 | Set [PV] Upper Range Value |
| 37 | Set [PV] Lower Range Value |
| 38 | Reset Configuration Changed Flag |
| 39 | EEPROM Control |
| 40 | Enter/Exit Fixed Current Mode |
| 41 | Perform Transmitter Self Test |
| 42 | Perform Master Reset |
| 43 | Set PV Zero |
| 44 | Write PV Units |
| 45 | Trim [PV Current] DAC Zeros |
| 46 | Trim [PV Current] DAC Gain |
| 47 | Write [PV] Transfer Function |
| 48 | Read Additional Transmitter Status |
| 49 | Write PC Sensor Serial Number |
| 50 | Read Dynamic Variable Assignments |
| 51 | Write Dynamic Variable Assignments |
| 52 | Set Transmitter Variable Zero |
| 53 | Write Transmitter Variable Units |
| 54 | Read Transmitter Variable Info |
| 55 | Write Transmitter Variable Damping Value |
| 56 | Write Transmitter Variable Sensor Serial No |
| 108 | Read All Dynamic Variables |

## 8.6 SNMP Client Protocol

The following function blocks are used to send SNMP request messages. These are used to retrieve the values of specific data objects in the remote device. Objects are specified by their object identifier (OID) which is a dotted numeric string eg '1.3.6.1.4.1.27982.1.1.1.1'.

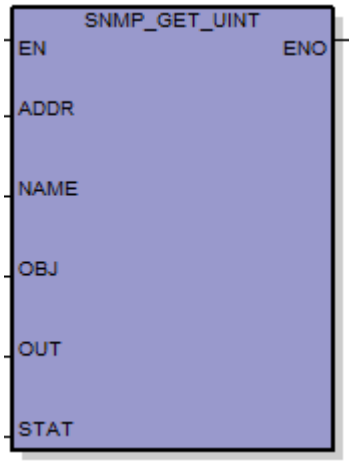
### 8.6.1 SNMP_GET_INT




Retrieves an integer object value from a remote device and stores it in a DINT variable.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) from which to read. |
| **NAME** | STRING(255) | The community name to use in the request message. |
| **OBJ** | STRING(255) | The object identifier in the remote device. |
| **OUT** | STRING(128) | Name of a DINT variable in which to store the retrieved value. |
| **STAT** | STRING(128) | Name of an integer variable in which to store a status value: 0 if success, non-zero if error. |

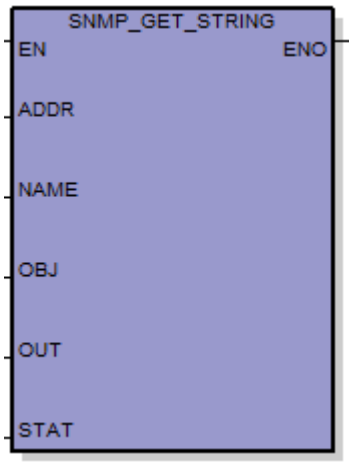
### 8.6.2 SNMP_GET_UINT




Retrieves an unsigned integer object value from a remote device and stores it in a DINT variable.

Parameters are as for SNMP_GET_INT.

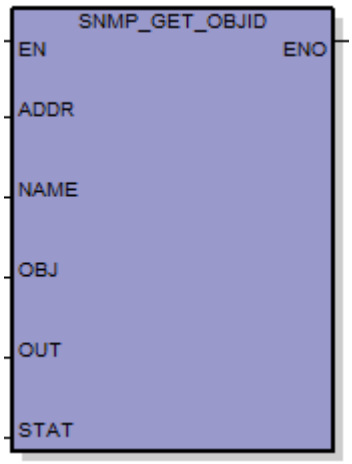
### 8.6.3 SNMP_GET_STRING




Retrieves a string object value from a remote device and stores it in a STRING variable.

Parameters are as for SNMP_GET_INT (except that *OUT* should be the name of a STRING variable).

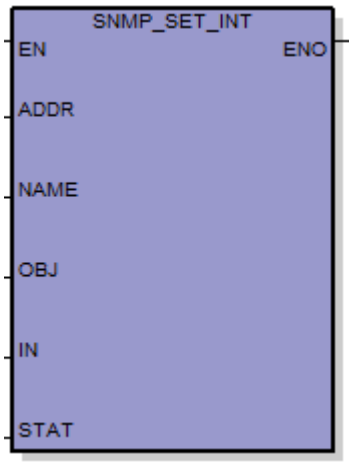
### 8.6.4 SNMP_GET_OBJID




Retrieves an OID object value from a remote device and stores it in a STRING variable.

Parameters are as for SNMP_GET_INT (except that *OUT* should be the name of a STRING variable).

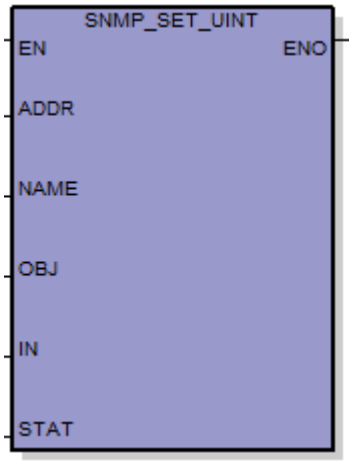
### 8.6.5 SNMP_SET_INT




Writes an integer value to a remote device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) from which to read. |
| **NAME** | STRING(255) | The community name to use in the request message. |
| **OBJ** | STRING(255) | The object identifier in the remote device. |
| **IN** | DINT | Value to write |
| **STAT** | STRING(128) | Name of an integer variable in which to store a status value: 0 if success, non-zero if error. |

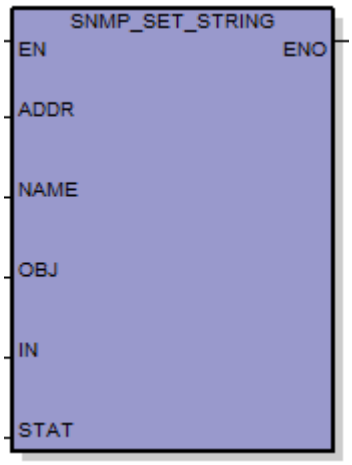
### 8.6.6 SNMP_SET_UINT




Writes an unsigned integer value to a remote device.

Parameters are as for SNMP_SET_INT (except that the *IN* parameter is now UDINT).

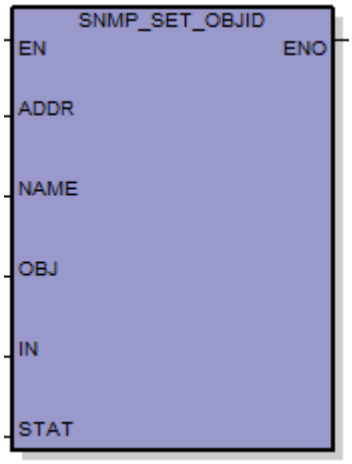
### 8.6.7 SNMP_SET_STRING




Writes a string value to a remote device.

Parameters are as for SNMP_SET_INT (except that the *IN* parameter is now STRING).

### 8.6.8 SNMP_SET_OBJID



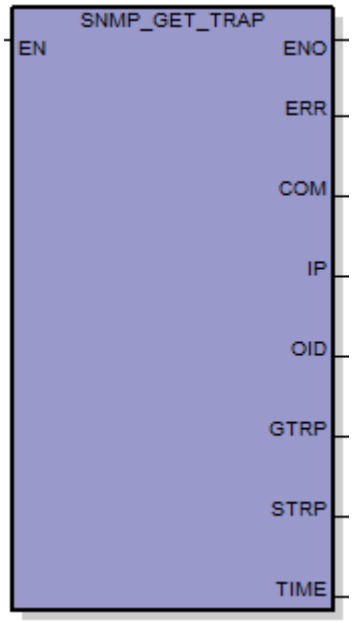

Writes an OID value to a remote device.

Parameters are as for SNMP_SET_INT (except that the *IN* parameter is now STRING).

## 8.7 SNMP Trap Protocol

The following function blocks are used to send or receive SNMP Trap messages (asynchronous notifications).

### 8.7.1 SNMP_GET_TRAP

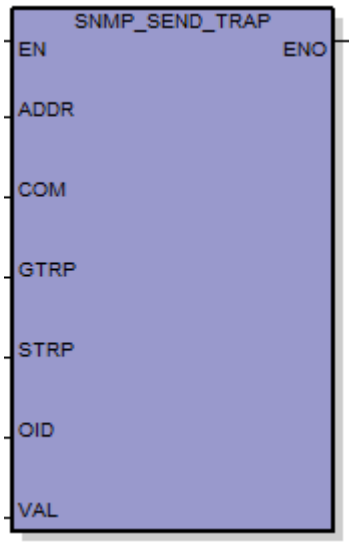



When a trap message is received, details associated with the message are stored in a queue. This function block returns details about the oldest trap message in the queue (if any).

| Parameter | Type | Description |
|---|---|---|
| ► **ERR** | INT | Status code. 0=message details successfully returned. Non-zero indicates that there is no trap message in the queue, or other error condition. |
| ► **COM** | STRING(128) | The community string specified in the trap message. Note: there is no validation of the authenticity of this community string for trap messages received. Empty string |
| ► **IP** | STRING(16) | The IP address of the agent that sent the trap message. |
| ► **OID** | STRING(255) | The object identifier associated with the trap message. |
| ► **GTRP** | UINT | The generic trap value of the trap message. The permissible values for this parameter are defined in *RFC 1157: A Simple Network Management Protocol (SNMP)*. |
| ► **STRP** | UINT | The specific trap value of the trap message. This value is device specific. |
| ► **TIME** | UDINT | Message timestamp (seconds since 1-Jan-1970) |

If no trap messages have been received then an empty string is returned for *COM*, *IP* and *OID*, and 0 is returned for *GTRP*, *STRP* and *TIME*.

### 8.7.2 SNMP_SEND_TRAP



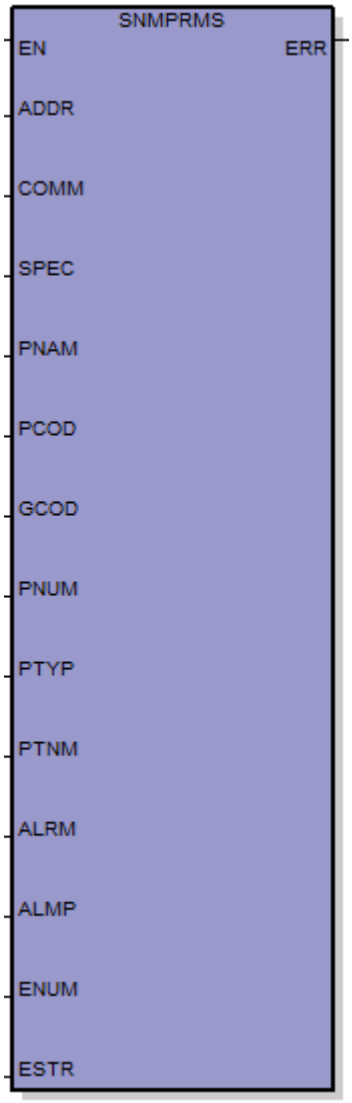

Sends a trap message containing the value of a particular SNMP object to a remote device.

When generating this message, the IP address of the default Ethernet port of the Kingfisher CP30 or G30 processor is used for the agent address and the current time is used for the time stamp parameter of the trap message.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | DINT | Address of the RTU (1-65520) to which to send trap message. |
| **COM** | STRING(128) | The community name to use in the trap message. |
| **GTRP** | UINT | The generic trap value to use in the trap message. The permissible values for this parameter are defined in *RFC 1157: A Simple Network Management Protocol (SNMP)*. |
| **STRP** | UINT | The specific trap value to use in the trap message. |
| **OID** | STRING(255) | The object identifier to use in the trap message. |
| **VAL** | DINT | Value of the object |

## 8.8 SNMP RMS Trap Protocol

### 8.8.1 SNMPRMS




Sends an SNMP Trap message to a remote device, emulating an *RMS Systems* RTU.

SNMP RMS messages differ from SNMP_SEND_TRAP messages in that information for multiple object identifiers defined in the RMS Systems MIB are sent within a single SNMP trap message. Object values can also be sent in the trap message.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) to which to send trap message. |
| **COM** | STRING(64) | The community name to use in the trap message. |
| **SPEC** | DINT | The specific RMS Systems message to send (101-111).<br>See Supported Message Types. |
| **PNAM** | STRING(20) | The port name associated with the SNMP trap.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.1 |
| **PCOD** | STRING(20) | The position code of the port associated with the SNMP trap.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.2 |
| **GCOD** | STRING(20) | The group code of the port associated with the SNMP trap.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.3 |
| **PNUM** | DINT | The overall port number associated with the SNMP trap.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.4 |
| **PTYP** | DINT | The port type (1-4) associated with the SNMP trap: 1=digital, 2=analog, 3=temperature, 4=virtual.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.5 |
| **PTNM** | DINT | The port type number of the port associated with the SNMP trap. This is the port number within the given port type rather than the overall port number within the configuration.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.6 |
| **ALRM** | DINT | The number of alarms on the given port.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.7 |
| **ALMP** | DINT | The alarm priority level of the port (1 = highest priority).<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.8 |
| **ENUM** | DINT | The external device number.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.9 |
| **ESTR** | STRING(20) | The external device string.<br>OID for this parameter is 1.3.6.1.4.1.4119.1.7.7.10 |
| ► **ERR** | BOOL | Set TRUE if an error occurs. |

### *Supported Message Types*

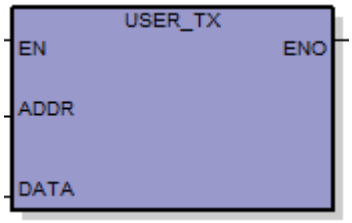
The following message types are supported:

| *SPEC* | Name | Description |
|---|---|---|
| 101 | rtuAlarm | Sent when a port goes into alarm |
| 102 | rtuARA | Sent when a port goes into ARA |
| 103 | rtuHistoric | Sent when a port goes into Historic |
| 104 | rtuNormal | Sent when a port goes normal |
| 105 | rtuOutputActivated | Sent when an output port is activated |
| 106 | rtuOutputDeactivated | Sent when an output port is deactivated |
| 107 | rtuPowerFail | Sent when power fails |
| 108 | rtuLogFull | Sent when a log reaches the specified alarm threshold |
| 109 | rtuUnitConfigChange | Sent when a unit setting has been changed |
| 110 | rtuPortConfigChange | Sent when a port setting has been changed |
| 111 | rtuRuleFail | Sent when the rules are enabled and they become invalid (either through a rule change or some other change in the RTU configuration) |

## 8.9 User Defined Protocol

The following function blocks are used to send and receive arbitrary communications messages. These can be used to implement simple protocols.

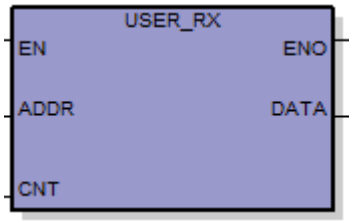
### 8.9.1 USER_TX




Sends up to 255 bytes to a remote device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) to which to send the data. |
| **DATA** | STRING(255) | Data bytes to send |

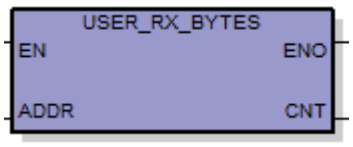
### 8.9.2 USER_RX




Reads up to 255 bytes from an internal buffer containing data received from a remote device.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) from which to read data. |
| **CNT** | UINT | Maximum number of bytes to read (1-255) |
| ► **DATA** | STRING(255) | Received data bytes. If no data have been received this will be an empty string. |

### 8.9.3 USER_RX_BYTES

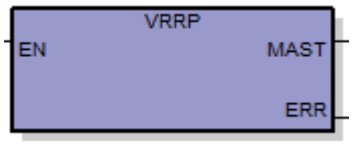



Returns the number of bytes that have been received from the remote device, but not yet read (using USER_RX).

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) to check for received data. |
| ► **CNT** | UINT | Number of bytes available to read. |

## 8.10 VRRP Protocol

### 8.10.1 VRRP



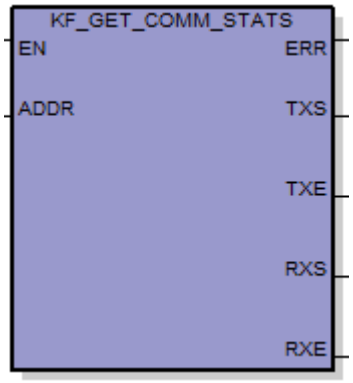

Returns the master/backup VRRP state for the local RTU.

| Parameter | Type | Description |
|---|---|---|
| ► **MAST** | BOOL | Set TRUE when the local RTU is operating as a master VRRP router.<br>Set FALSE when the local RTU is operating as a backup VRRP router. |
| ► **ERR** | BOOL | Set TRUE if there is an error. |

## 8.11 General Communications

These function blocks return information about the status of the various communications links managed by the RTU.

### 8.11.1 KF_GET_COMM_STATS




Returns communications statistics for the link between the local RTU and a particular remote RTU (or all RTUs).

See Interpreting communications statistics for more details on the meanings of the various statistics.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) for which to return communications statistics. Set to 0 to return statistics for all remote RTUs. |
| ► **ERR** | BOOL | TRUE if an error occurred, or if no communications with the selected RTU have been attempted. |
| ► **TXS** | UDINT | Number of messages successfully transmitted to the selected RTU (or all RTUs if *ADDR*=0). |
| ► **TXE** | UDINT | Number of messages that were not able to be transmitted to the selected RTU (or all RTUs if *ADDR*=0). |
| ► **RXS** | UDINT | Number of messages successfully received from the selected RTU (or all RTUs if *ADDR*=0). |
| ► **RXE** | UDINT | Number of messages for which a valid response was not received from the selected RTU (or all RTUs if *ADDR*=0) |

#### *Interpreting communications statistics*

The meaning of the various statistics varies somewhat depending on the protocol and whether the RTU is operating as a slave or a master.

When the RTU is a slave:

- RXS is the number of valid messages received.
- RXE is not used. Invalid messages and messages not addressed to the RTU are ignored.
- TXS is the number of messages sent. For Modbus, this includes exception responses (e.g. where the requested data are not available).
- TXE is the number of reply messages that could not be sent, or DNP3 unsolicited messages where the RTU cannot connect to master.

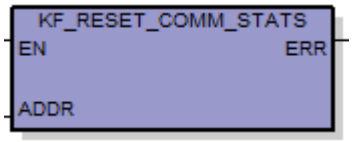
When the RTU is a master:

- TXS is the number of messages sent.
- TXE is the number of times where it was not possible to connect to the slave (Ethernet only). It also includes cases where there is no route defined for the slave address (except Modbus protocol).
- RXS is the number of valid messages received.
- RXE is the number of times an exception/error response was received, or where no response was received within the timeout period.

Furthermore:

- If a retry count is set then each attempt counts as a message.
- For DNP3, the message counts are really message fragment counts.

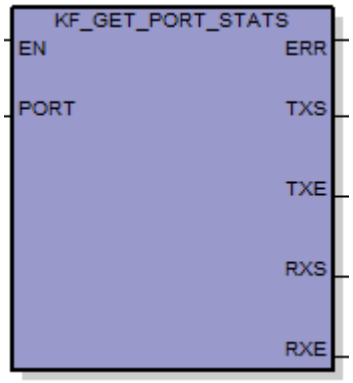
### 8.11.2 KF_RESET_COMM_STATS




Resets communications statistics for the link between the local RTU and a particular remote RTU (or all RTUs).

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) for which to reset communications statistics. Set to 0 to reset statistics for all remote RTUs. |
| ► **ERR** | BOOL | TRUE if an error occurred, or if no communications to the selected RTU have been attempted. |

### 8.11.3 KF_GET_PORT_STATS



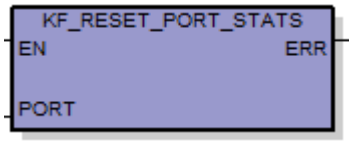

Returns communications statistics for a physical RTU port.

See Interpreting communications statistics for more details on the meanings of the various statistics.

| Parameter | Type | Description |
|---|---|---|
| **PORT** | STRING(8) | Port identifier. This is a string of the form '*s*:*p*', where *s* is the slot number (1-64, or 0 for the active CP30) and *p* is the port number (1-3, port 1 is the topmost port). |
| ► **ERR** | BOOL | TRUE if an error occurred, or if no communications on the selected port have been attempted. |
| ► **TXS** | UDINT | Number of messages successfully transmitted on the selected port. |
| ► **TXE** | UDINT | Number of messages that were not able to be transmitted on the selected port. |
| ► **RXS** | UDINT | Number of messages successfully received on the selected port. |
| ► **RXE** | UDINT | Number of messages sent on the selected port for which a valid response was not received. |

For example, to retrieve statistics for the built-in Ethernet on the CP30 you would set *PORT*='0:1'. To specify port 3 on an MC31 in slot 22, use *PORT*='22:3'.

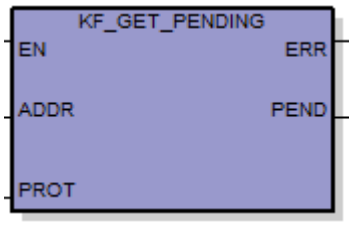
### 8.11.4 KF_RESET_PORT_STATS




Resets communications statistics for the specified port

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | STRING(8) | Port identifier. This is a string of the form '*s*:*p*', where *s* is the slot number (1-64, or 0 for the active CP30) and *p* is the port number (1-3, port 1 is the topmost port). |
| ► **ERR** | BOOL | TRUE if an error occurred, or if no communications on the selected port have been attempted. |

### 8.11.5 KF_GET_PENDING

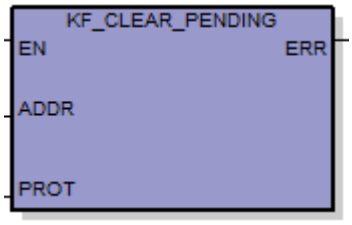



Indicates whether a message is pending (awaiting reply), either for a particular remote RTU, or a particular protocol, or globally.

This is typically used to prevent further messages being initiated until the previous one has completed or timed out.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) to check for pending messages. Set to 0 to check for pending messages for all remote RTUs. |
| **PROT** | USINT | Protocol to check for pending messages. Set to 0 to check for pending messages for all protocols. See ISaGRAF Constants for a list of defined words that may be used to specify the protocol.. |
| ► **ERR** | BOOL | TRUE if an error occurred. |
| ► **PEND** | BOOL | TRUE if there is an outstanding message for which no reply has been received, for the specified remote RTU address and/or protocol. |

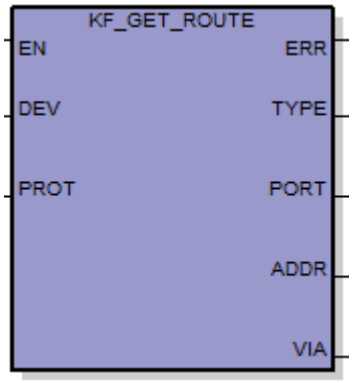
### 8.11.6 KF_CLEAR_PENDING




Clears the message pending flag, either for a particular remote RTU, or a particular protocol, or globally.

| Parameter | Type | Description |
|---|---|---|
| **ADDR** | UINT | Address of the RTU (1-65520) for which to clear the pending flag. Set to 0 to clear the flag for all remote RTUs. |
| **PROT** | USINT | Protocol for which to clear the pending flag. Set to 0 to clear the flag for all protocols. See ISaGRAF Constants for a list of defined words that may be used to specify the protocol.. |
| ► **ERR** | BOOL | TRUE if an error occurred. |

### 8.11.7 KF_GET_ROUTE

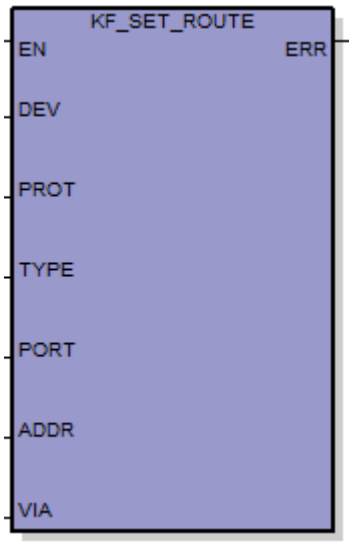



Returns information about a communications route. A route specifies the port (and possible intermediate RTU) to use in order to send a message to a particular RTU address using a particular protocol.

Routes may be created in Toolbox PLUS+, or may be added dynamically (e.g when a slave RTU is polled by a master), or may be modified in logic (using KF_SET_ROUTE).

| Parameter | Type | Description |
|---|---|---|
| **DEV** | UINT | Address of the RTU (1-65520) for which route information is to be returned. |
| **PROT** | USINT | Protocol for which route information is to be returned. See ISaGRAF Constants for a list of defined words that may be used to specify the protocol.. |
| ► **ERR** | BOOL | TRUE if an error occurred. |
| ► **TYPE** | DINT | Returns type of route: 0=direct (messages can be sent directly to the destination RTU), 1=indirect (messages must be sent via an intermediate RTU). |
| ► **PORT** | STRING(8) | For direct routes, returns the port to use to communicate with the destination RTU. This is a string of the form '*s*:*p*', where *s* is the slot number (1-64, or 0 for the active CP30) and *p* is the port number (1-3, port 1 is the topmost port). |
| ► **ADDR** | STRING(16) | For direct routes using an Ethernet port, returns IP address of destination RTU, e.g. '192.168.0.42' |
| ► **VIA** | UINT | For indirect routes, returns the address (1-65520) of the intermediate RTU through which messages are to be sent. |

### 8.11.8 KF_SET_ROUTE




Allows an existing communications route to be modified.

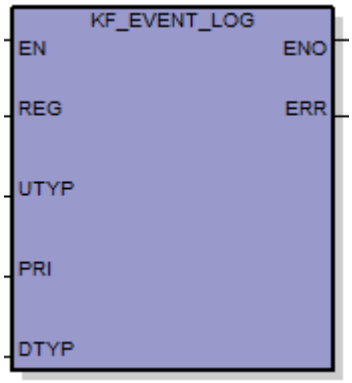
This can be used to change the port that is used to communicate with a particular RTU – for example, if a fault is detected.

| Parameter | Type | Description |
|---|---|---|
| **DEV** | UINT | Address of the RTU (1-65520) for which route information is to be modified. |
| **PROT** | USINT | Protocol for which route information is to be modified. See ISaGRAF Constants for a list of defined words that may be used to specify the protocol.. |
| **TYPE** | DINT | Sets type of route: 0=direct (messages can be sent directly to the destination RTU), 1=indirect (messages must be sent via an intermediate RTU).<br>The defined words ROUTE_DIRECT and ROUTE_INDIRECT may also be used. |
| **PORT** | STRING(8) | For direct routes, sets the port to use to communicate with the destination RTU. This is a string of the form '*s*:*p*', where *s* is the slot number (1-64, or 0 for the active CP30) and *p* is the port number (1-3, port 1 is the topmost port). |
| **ADDR** | STRING(16) | For direct routes using an Ethernet port, sets IP address of destination RTU, e.g. '192.168.0.42' |
| **VIA** | UINT | For indirect routes, sets the address (1-65520) of the intermediate RTU through which messages are to be sent. |
| ► **ERR** | BOOL | TRUE if an error occurred. |

## 8.12 Event Logging

The following function blocks are used to create and manage event logs on the local RTU, or a remote RTU.

### 8.12.1 KF_EVENT_LOG



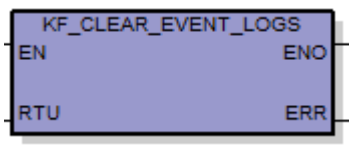

Logs the current value of any variable along with timestamp and other information.

Timestamps are logged accurate to one second, except for variables mapped to DI-10 or IOD-MX2/3/4 SOE inputs, which can be logged accurate to within a few milliseconds.

Class 1/2/3 DNP3 variables will be automatically logged whenever their value changes. However this function block can be used to create additional log entries by logging variables even when they haven't changed (e.g. periodically).

| Parameter | Type | Description |
|---|---|---|
| **REG** | STRING(16) | Name of variable to log, e.g. 'DNPAI3' or 'SL05IO3DI1'. Note that for IOPOINT variables (DNP3 or I/O module points) it is not necessary to include the '.value' suffix. |
| **UTYP** | USINT | For DNP3 variables, this is the event variation (1-4).<br>For other variables, this is a user type code (0-31) which can optionally be used to filter events when uploading. |
| **PRI** | USINT | For DNP3 variables, this is the event class (1-3).<br>For other variables, this is a user priority code (0-7) which can optionally be used to filter events when uploading. |
| **DTYP** | USINT | Not used (set to 0) |
| ► **ERR** | DINT | 0 if OK, non-zero if an error occurred. |

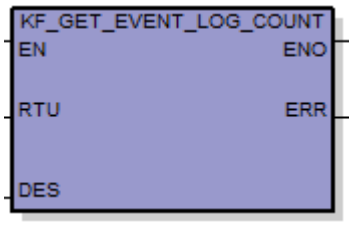
### 8.12.2 KF_CLEAR_EVENT_LOGS




Clears all event logs on the local or a remote RTU.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of remote RTU (1-65520) on which to clear event logs. Set to 0 to clear event logs on the local RTU. |
| ► **ERR** | DINT | 0 if OK, non-zero if an error occurred. |

### 8.12.3 KF_GET_EVENT_LOG_COUNT

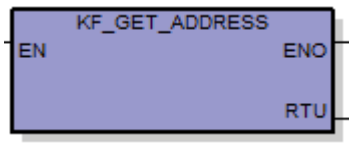



Returns the number of event logs currently stored on the local or a remote RTU.

| Parameter | Type | Description |
|---|---|---|
| **RTU** | UINT | Address of remote RTU (1-65520) from which to obtain event log count. Set to 0 to return the number of event logs on the local RTU. |
| **DES** | STRING(16) | Name of an integer variable in which to store the event log count. |
| ► **ERR** | DINT | 0 if OK, non-zero if an error occurred. |

## 8.13 RTU System Data

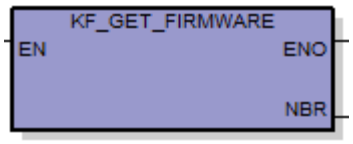
### 8.13.1 KF_GET_ADDRESS




Returns the address of the local RTU.

| Parameter | Type | Description |
|---|---|---|
| ► **RTU** | UINT | Address of local RTU (1-65520) |

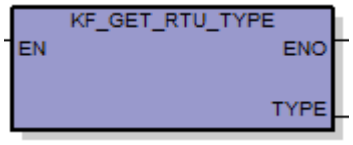
### 8.13.2 KF_GET_FIRMWARE




Returns the local RTU firmware version.

| Parameter | Type | Description |
|---|---|---|
| ► **NBR** | UDINT | Firmware version number |

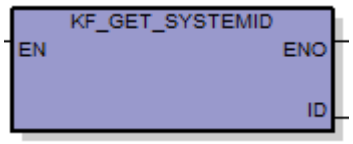
### 8.13.3 KF_GET_RTU_TYPE




Identifies the type of processor module (CP30 or G30)

| Parameter | Type | Description |
|---|---|---|
| ► **TYPE** | USINT | Processor type: 1=CP30, 2=G30 |

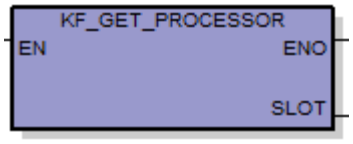
### 8.13.4 KF_GET_SYSTEMID




Returns the configured system ID byte (Kingfisher protocol message prefix) for the local RTU.

| Parameter | Type | Description |
|---|---|---|
| ► **ID** | INT | System ID byte (normally 174) |

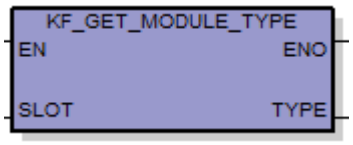
### 8.13.5 KF_GET_PROCESSOR




Identifies the backplane slot containing the active processor module.

| Parameter | Type | Description |
|---|---|---|
| ► **SLOT** | UINT | Active processor slot number (1-64), or 0 for G30. |

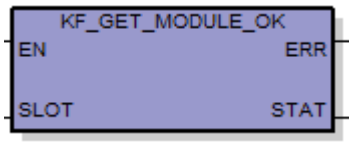
### 8.13.6 KF_GET_MODULE_TYPE




Identifies the module type installed in the specified backplane slot.

| Parameter | Type | Description |
|---|---|---|
| **SLOT** | UINT | Slot number (1-64) |
| ► **TYPE** | UINT | Detected module type (1-255):<br>1=AI-1/AI-4, 2=AO-2, 5=DI-1, 6=DI-5, 7=DO-1, 8=DO-2/DO-5/DO-6, 9=DI-10, 11=IO-2, 12=IO-3, 14=IO-4, 15=AO-3, 19=AI-10, 31=PS-1x/PS-2x, 48=MC-30, 50=IO-5, 60=CP-30, 88=MC-31, 255=No module installed |

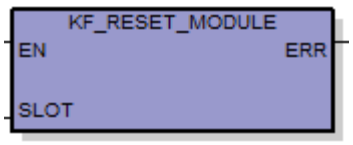
### 8.13.7 KF_GET_MODULE_OK




Compares the module detected in a backplane slot with the module configured in the RTU's configuration.

| Parameter | Type | Description |
|---|---|---|
| **SLOT** | UINT | Slot number (1-64) |
| ► **ERR** | BOOL | Set TRUE if the specified slot number is invalid. |
| ► **STAT** | BOOL | Module status. Set TRUE if a module is present in the specified slot, and its type matches that specified in the RTU's configuration. |

### 8.13.8 KF_RESET_MODULE



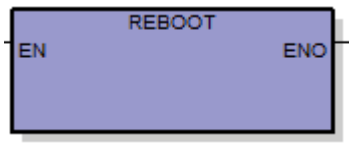

Reset/reconfigure the module in the specified backplane slot.

For MC30/MC31 modules, a full reboot will be performed, after which the module will be re-detected and its configuration reloaded.

For I/O modules, the CP30 internally marks the module as absent, after which the module will be re-detected and its configuration reloaded.

| Parameter | Type | Description |
|---|---|---|
| **SLOT** | USINT | Slot number (1-64) |
| ► **ERR** | BOOL | Set TRUE if the specified slot number is invalid. |

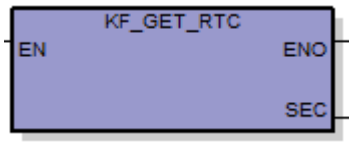
### 8.13.9 REBOOT




Reboots the local processor module.

Note: This function block has no input or output parameters, so it is not directly usable in Function Block Diagram (FBD) programs. You can, however, create a small Ladder Diagram function that calls REBOOT, then call that function from a FBD program.

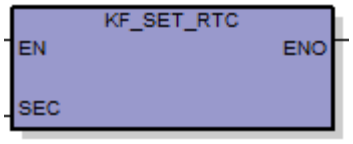
### 8.13.10 KF_GET_RTC




Returns the current RTU date/time, expressed as a single integer.

| Parameter | Type | Description |
|---|---|---|
| ► **SEC** | DINT | Current RTU time: seconds since 0:00 1-Jan-1970 |

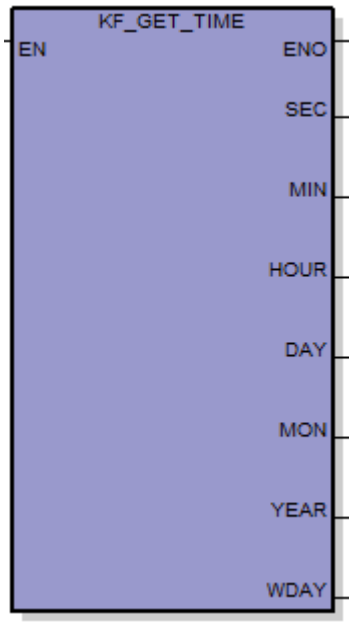
### 8.13.11 KF_SET_RTC




Sets the current RTU date/time, expressed as a single integer.

| Parameter | Type | Description |
|---|---|---|
| **SEC** | DINT | New RTU time: seconds since 0:00 1-Jan-1970 |

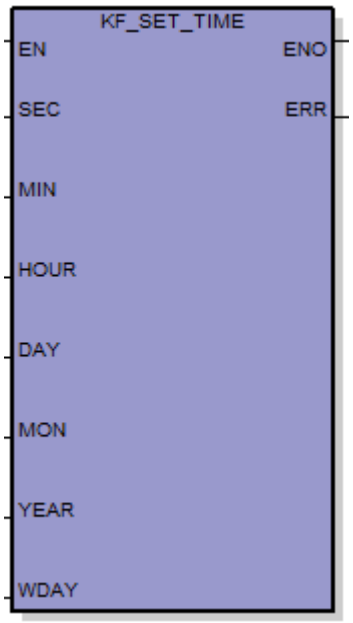
### 8.13.12 KF_GET_TIME




Returns the current RTU date/time, expressed as separate time components

| Parameter | Type | Description |
|---|---|---|
| ► **SEC** | DINT | Seconds (0-59) |
| ► **MIN** | DINT | Minutes (0-59) |
| ► **HOUR** | DINT | Hours (0-23) |
| ► **DAY** | DINT | Day of month (1-31) |
| ► **MON** | DINT | Month (1-12) |
| ► **YEAR** | DINT | Years since 1900 (0-170) |
| ► **WDAY** | DINT | Day of week (1=Sun, 7=Sat) |

### 8.13.13 KF_SET_TIME



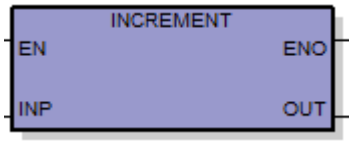

Sets the current RTU date/time, expressed as separate time components

| Parameter | Type | Description |
|---|---|---|
| **SEC** | DINT | Seconds (0-59) |
| **MIN** | DINT | Minutes (0-59) |
| **HOUR** | DINT | Hours (0-23) |
| **DAY** | DINT | Day of month (1-31) |
| **MON** | DINT | Month (1-12) |
| **YEAR** | DINT | Years since 1900 (0-170) |
| **WDAY** | DINT | Ignored (may be set to any value 1-7) |
| ► **ERR** | DINT | Returns non-zero if parameter error |

## 8.14 Maths and Logic

The following function blocks provide a few additional capabilities beyond those provided by the standard ISaGRAF library.

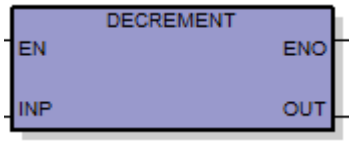
### 8.14.1 INCREMENT




Adds one to an integer variable.

| Parameter | Type | Description |
|---|---|---|
| **INP** | DINT | Input variable |
| **► OUT** | DINT | Output variable. May be the same as the input variable. |

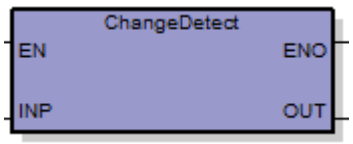
### 8.14.2 DECREMENT




Subtracts one from an integer variable.

| Parameter | Type | Description |
|---|---|---|
| **INP** | DINT | Input variable |
| **► OUT** | DINT | Output variable. May be the same as the input variable. |

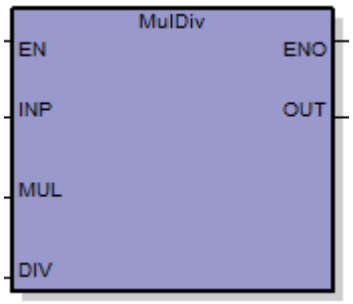
### 8.14.3 ChangeDetect




Detects change in an integer variable.

| Parameter | Type | Description |
|---|---|---|
| **INP** | DINT | Input variable |
| **► OUT** | DINT | Set to 1 if the input value has changed since the last time the function block was executed, 0 otherwise |

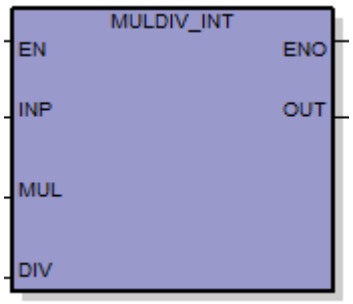
### 8.14.4 MulDiv




Scales a floating point variable by multiplying it by an integer value then dividing by another value. The result is a floating point value.

| Parameter | Type | Description |
|---|---|---|
| **INP** | REAL | Input variable |
| **MUL** | DINT | Multiplier |
| **DIV** | DINT | Divisor |
| **► OUT** | REAL | Output variable. May be the same as the input variable. |

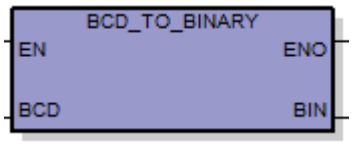
### 8.14.5 MULDIV_INT




Scales an integer variable by multiplying it by a value then dividing by another value. The result is an integer.

| Parameter | Type | Description |
|---|---|---|
| **INP** | DINT | Input variable |
| **MUL** | DINT | Multiplier |
| **DIV** | DINT | Divisor |
| **► OUT** | DINT | Output variable. May be the same as the input variable. |

### 8.14.6 BCD_TO_BINARY

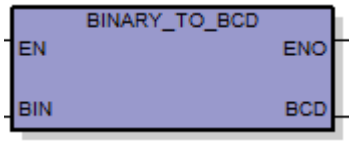



Converts a variable from Binary Coded Decimal to a normal integer value.

BCD uses each group of 4 bits to represent a decimal digit.

| Parameter | Type | Description |
|---|---|---|
| **BCD** | DINT | Input variable (16#000000-16#999999) |
| ► **BIN** | DINT | Output variable (0-999999). May be the same as the input variable. |

### 8.14.7 BINARY_TO_BCD

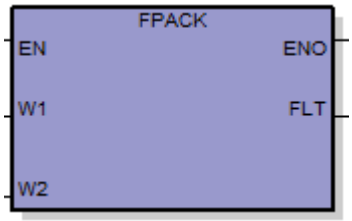



Converts an integer variable to Binary Coded Decimal.

BCD uses each group of 4 bits to represent a decimal digit.

| Parameter | Type | Description |
|---|---|---|
| **BIN** | DINT | Input variable (0-999999) |
| ► **BCD** | DINT | Output variable (16#000000-16#999999). May be the same as the input variable. |

### 8.14.8 FPACK

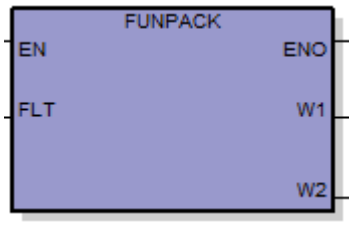



Converts two 16-bit integer values to a floating point value.

Typically used when reading floating point values using the Modbus protocol, which splits floating point values across two 16-bit registers.

| Parameter | Type | Description |
|---|---|---|
| **W1** | UINT | First integer variable |
| **W2** | UINT | Second integer variable |
| ► **FLT** | REAL | Floating point output variable |

### 8.14.9 FUNPACK

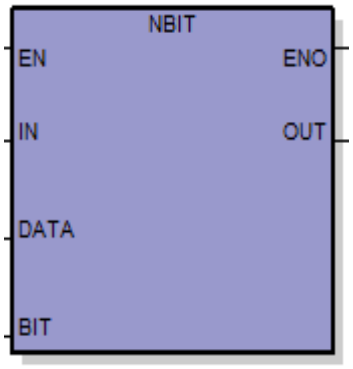



Converts a floating point value into two 16-bit integers.

Typically used when writing floating point values using the Modbus protocol, which splits floating point values across two 16-bit registers.

| Parameter | Type | Description |
|---|---|---|
| **FLT** | REAL | Floating point variable |
| **► W1** | UINT | First integer output variable |
| **► W2** | UINT | Second integer output variable |

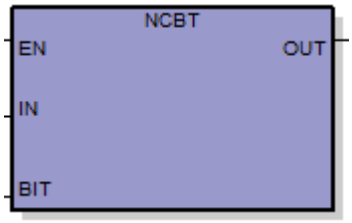
### 8.14.10 NBIT




Set/clear a single bit in an integer variable.

| Parameter | Type | Description |
|---|---|---|
| **IN** | UINT | Input variable |
| **DATA** | BOOL | TRUE=set bit, FALSE=clear bit |
| **BIT** | USINT | Bit number to set/clear (1-16) |
| **► OUT** | UINT | Output variable. May be the same as the input variable. |

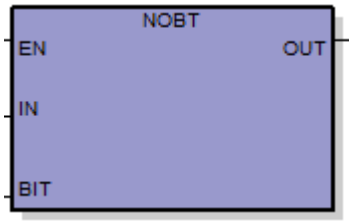
### 8.14.11 NCBT




Tests whether a single bit in an integer variable is closed (set).

| Parameter | Type | Description |
|---|---|---|
| **IN** | UINT | Input variable |
| **BIT** | USINT | Bit number to test (1-16) |
| ► **OUT** | BOOL | TRUE if bit is set (logic 1). |

### 8.14.12 NOBT




Tests whether a single bit in an integer variable is open (clear).

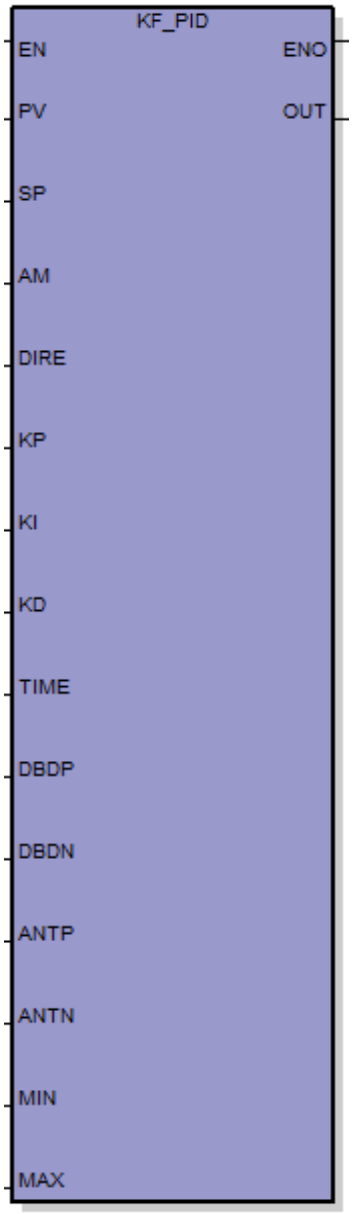
| Parameter | Type | Description |
|---|---|---|
| **IN** | UINT | Input variable |
| **BIT** | USINT | Bit number to test (1-16) |
| ► **OUT** | BOOL | TRUE if bit is clear (logic 0). |

## 8.15 PID Controller

The proportional-integral-derivative (PID) function block calculates the difference (or error value) between a measured process value and a desired setpoint and attempts to minimise this difference by adjusting the output variable. The PID calculation takes into account:

- A proportional component which calculates a response proportional to the current error value
- An integral component which calculates a response based upon accumulated error
- A derivative component which calculates a response based upon the rate of change of the error value.

### 8.15.1 KF_PID




Given a measured process value and a desired set-point, calculates the appropriate output value based on a Proportional-Integral-Derivative control formula.

The following formula is used by the PID function block:

$$K_p e_n + K_i \int e + K_d \frac{1}{6t}\left(e_n - e_{(n-3)} + 3e_{(n-1)} - 3e_{(n-2)}\right)$$

Evaluated from 0 to *t*, where:

$K_p$ = Proportional constant

$K_i$ = Integral constant

$K_d$ = Derivative constant

*t* = Sample interval

$e_n$ = Process error for current sample

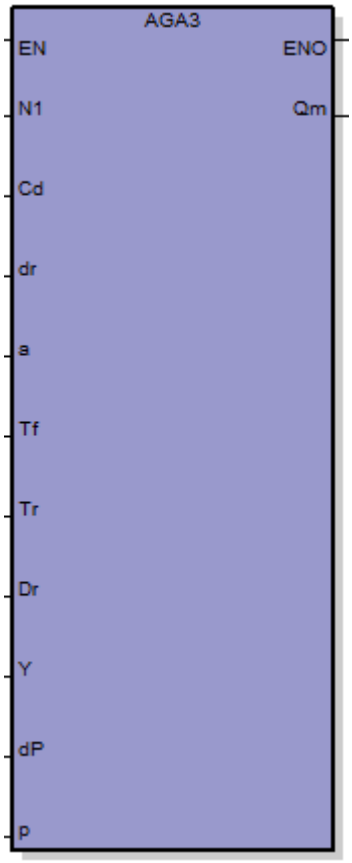
$e_{n-1}$ = Process error for previous sample, etc.

| Parameter | Type | Description |
|---|---|---|
| **PV** | REAL | Measured process value |
| **SP** | REAL | Desired set-point for the process value |
| **AM** | BOOL | Automatic/manual mode.<br>If TRUE (automatic mode), the function block will calculate an output value based on proportional, integral and derivative components of the error value. If FALSE (manual mode), the function block output value will simply track the set-point value without calculation. |
| **DIRE** | BOOL | Direct/reverse mode.<br>If TRUE (direct mode), the output value will increase as the measured process value increases. If FALSE (reverse mode), the output value will decrease as the measured process value increases. |
| **KP** | REAL | PID proportional constant. |
| **KI** | REAL | PID integral constant.<br>This constant is set to zero when *PV* falls within the range of the anti-reset parameters, in order to prevent output value overshoot during initial tuning operations. |
| **KD** | REAL | PID derivative constant. |
| **TIME** | TIME | Time interval (ms) between sampling the process variable.<br>Note that the minimum interval that can be used is defined by the ISaGRAF sample time. |
| **DBDP** | REAL | Dead-band positive<br>The allowable positive error between *PV* and *SP*. If the positive error is less than this value, the output will not be adjusted. The dead-band settings can be used to suppress output hunting (oscillating output). |
| **DBDN** | REAL | Dead-band negative<br>The allowable negative error between *PV* and *SP*. If the negative error is less than this value, the output will not be adjusted. |
| **ANTP** | REAL | Anti-reset positive<br>The value of *PV* above which to suppress an integral response. The anti-reset range can be used suppress over-correction (and overshoot) of the output value during initial tuning actions where there is a significant difference between the *PV* and *SP*.<br>Set to 0.0 to disable the anti-reset feature. |
| **ANTN** | REAL | Anti-reset negative<br>The value of *PV* below which to suppress an integral response. |
| **MIN** | REAL | Minimum allowable output value. |
| **MAX** | REAL | Maximum allowable output value. |
| **► OUT** | REAL | Output value. |

## 8.16 Gas Flow Calculations

These function blocks perform various standard American Gas Association (AGA) and Chinese (GBT) gas calculations.

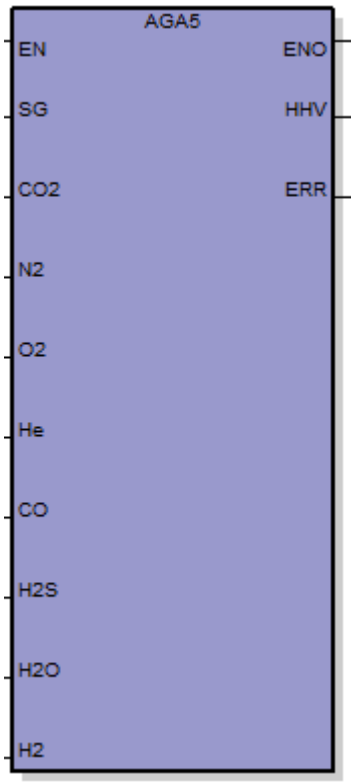
### 8.16.1 AGA3




Uses the factors method of the 1992 revision of the American Gas Association AGA-3 report to calculate the natural gas mass flow.

| Parameter | Type | Description |
|---|---|---|
| **N1** | REAL | Unit Conversion Factor (Orifice Flow) |
| **Cd** | REAL | Coefficient of Discharge - Orifice Plate |
| **dr** | REAL | Reference Orifice Plate Bore Diameter at reference remperature |
| **a** | REAL | Linear Coefficient of Thermal Expansion |
| **Tf** | REAL | Temperature |
| **Tr** | REAL | Reference Temperature |
| **Dr** | REAL | Reference meter tube, internal diameter at reference temperature |
| **Y** | REAL | Expansion Factor |
| **dP** | REAL | Differential Pressure |
| **p** | REAL | Density of Fluid at Flowing Conditions |
| **► Qm** | REAL | Calculated Mass Flow |

### 8.16.2 AGA5

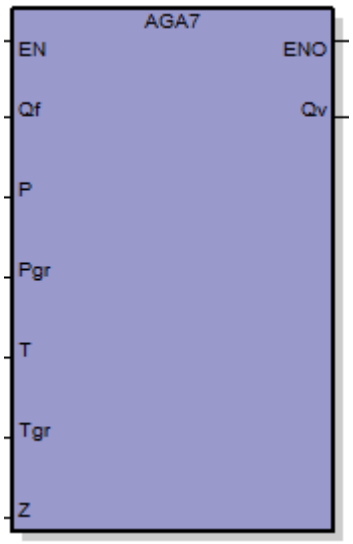



This calculation is used for gas energy metering applications and permits the calculation of Higher Heating Value (HHV, also known as the gross calorific value or gross energy) based upon volume flow consistent with AGA Report number 5 *Natural Gas Energy Measurement*.

This function accepts the volume percentage of gas constituents and returns the corresponding higher heating value in BTU.

| Parameter | Type | Description |
|---|---|---|
| **SG** | REAL | Specific gravity of the component gas under contract conditions. |
| **CO2** | REAL | Percentage volume of carbon dioxide |
| **N2** | REAL | Percentage volume of nitrogen |
| **O2** | REAL | Percentage volume of oxygen |
| **He** | REAL | Percentage volume of helium |
| **CO** | REAL | Percentage volume of carbon monoxide |
| **H2S** | REAL | Percentage volume of hydrogen sulphide |
| **H2O** | REAL | Percentage volume of water |
| **H2** | REAL | Percentage volume of hydrogen |
| ► **HHV** | REAL | Calculated Higher heating value in BTU |
| ► **ERR** | BOOL | Set TRUE if one or more parameters out of range |

### 8.16.3 AGA7




Uses the calculations from the 1980 revision of the American Gas Association AGA-7 report to calculate the volumetric flow of gas using the formula:

$$Q_v = \frac{Q_f \times P}{P_{gr} \times T_{gr}} \over T \times Z$$

where

$Q_v$ = Flow Volume
$Q_f$ = Flow Rate
$P$ = Measured Flow
$T$ = Measured Temperature
$P_{gr}$ = Reference Pressure
$T_{gr}$ = Reference Temperature
$Z$ = Gas Compressibility

| Parameter | Type | Description |
|---|---|---|
| **Qf** | REAL | Flow Rate at flowing conditions |
| **P** | REAL | Pressure |
| **Pgr** | REAL | Reference Pressure at specific gravity |
| **T** | REAL | Temperature |
| **Tgr** | REAL | Reference Temperature at specific gravity |
| **Z** | REAL | Compressibility Factor (output from AGA8 Gross calculation) |
| ► **Qv** | REAL | Calculated Volumetric Flow |

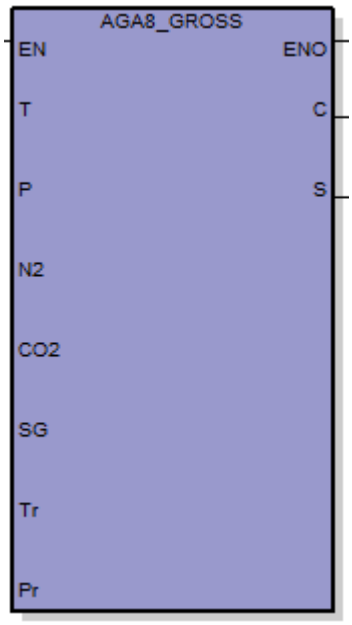
*Calculation Units*

There are no required units for this calculation. Users must take care in ensuring that the units used are appropriate as recommended below.

- Pressure and Reference Pressure must be in the same units and they must be absolute units of pressure, such as psia, kPaa, MPaa or similar. Absolute units for the line pressure are typically the pressure from the transmitter (in gauge units) plus the local atmospheric pressure or the reading from an absolute pressure transmitter.
- Temperature and Reference Temperature must be in the same units and they must be the absolute units of Kelvin (deg C + 273.15) or Rankine (deg F + 459.67).
- The Compressibility correction factor is the ratio of the compressibility of the gas at base conditions to the compressibility of the gas at flowing conditions. To calculate this, you need to execute two AGA8 calculations, using the same gas composition, but the first with the base pressure and temperature to calculate the base compressibility ($Z_b$) and the second with the flowing pressure and temperature to calculate the flowing compressibility ($Z_f$). The Compressibility correction factor is then $Z_b$ / $Z_f$. Either of the AGA8 Gross or AGA8 Detailed function blocks can be used, according to the available gas composition information available.

- Note that the Super-compressibility factor, $F_{pv}$ is the square root of the Compressibility correction Factor, and can be used ($Z = F_{pv}^2$) if it is supplied from an external source.
- The output of the function block, $Q_v$ will be in the same units as the supplied $Q_f$.

### 8.16.4 AGA8_GROSS

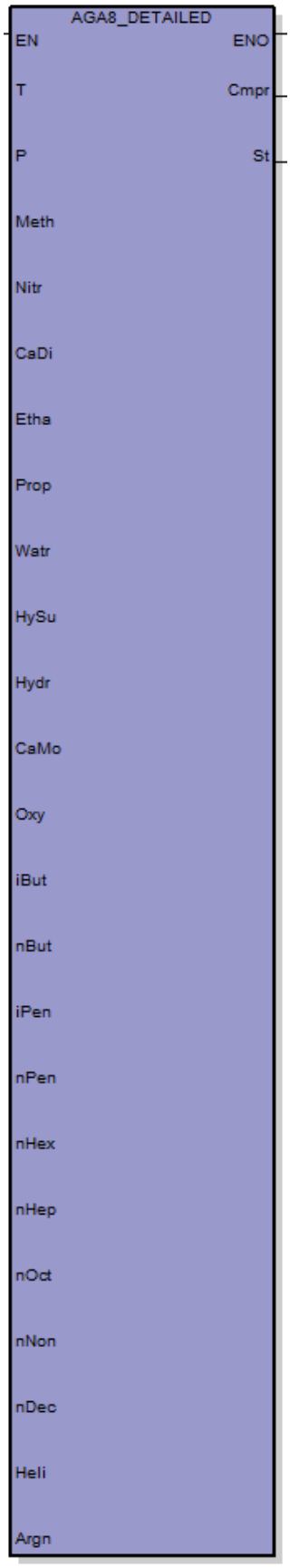



Uses the American Gas Association AGA-8 report for calculating gas compressibility using the gross method.

This implementation is based entirely on *Compressibility Factors of Natural Gas and Other Related Hydrocarbon Gases*, AGA Transmission Measurement Committee Report No. 8, Second Edition, November 1992.

| Parameter | Type | Description |
|---|---|---|
| **T** | REAL | Temperature (-8 – 62 °C) |
| **P** | REAL | Pressure (0-12 MPa) |
| **N2** | REAL | Molar fraction of nitrogen |
| **CO2** | REAL | Molar fraction of carbon dioxide |
| **SG** | REAL | Specific gravity |
| **Tr** | REAL | Reference temperature (°C) |
| **Pr** | REAL | Reference pressure (MPa) |
| **► C** | REAL | Calculated Compressibility factor |
| **► S** | INT | Status register: 0=OK, bits 0-2 indicate errors:<br>bit0=calculation error, bit1=pressure out of range, bit2=temperature out of range. |

### 8.16.5 AGA8_DETAILED




Uses the American Gas Association AGA-8 report for calculating gas compressibility using the detailed characterisation method.

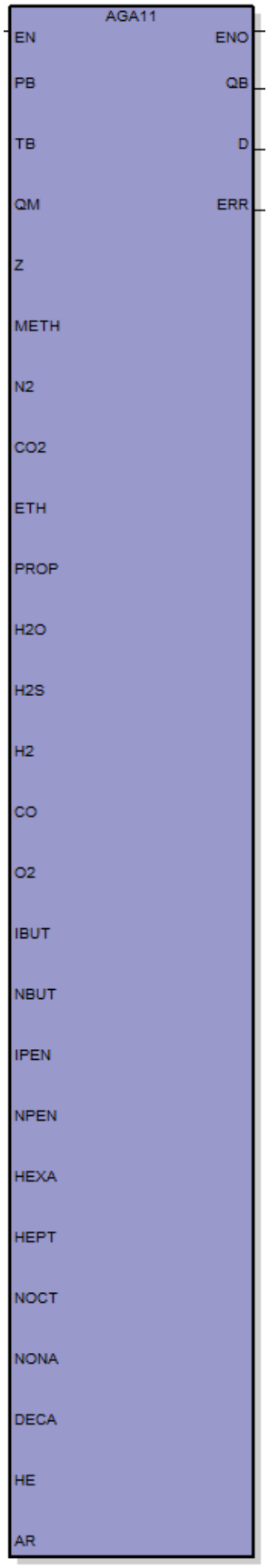
This implementation is based entirely on *Compressibility Factors of Natural Gas and Other Related Hydrocarbon Gases*, AGA Transmission Measurement Committee Report No. 8, Second Edition, November 1992.

| Parameter | Type | Description |
|---|---|---|
| **T** | REAL | Temperature (-130 – 400 °C) |
| **P** | REAL | Pressure (0-280 MPa) |
| **Meth** | REAL | Molar fraction of methane |
| **Nitr** | REAL | Molar fraction of nitrogen |
| **CaDi** | REAL | Molar fraction of carbon dioxide |
| **Etha** | REAL | Molar fraction of ethane |
| **Prop** | REAL | Molar fraction of propane |
| **Watr** | REAL | Molar fraction of water |
| **HySu** | REAL | Molar fraction of hydrogen sulphide |
| **Hydr** | REAL | Molar fraction of hydrogen |
| **CaMo** | REAL | Molar fraction of carbon monoxide |
| **iBut** | REAL | Molar fraction of i-butane |
| **nBut** | REAL | Molar fraction of n-butane |
| **iPen** | REAL | Molar fraction of i-pentane |
| **nPen** | REAL | Molar fraction of n-pentane |
| **nHex** | REAL | Molar fraction of n-hexane |
| **nHep** | REAL | Molar fraction of n-heptane |
| **nOct** | REAL | Molar fraction of n-octane |
| **nNon** | REAL | Molar fraction of n-nonane |
| **nDec** | REAL | Molar fraction of n-decane |
| **Heli** | REAL | Molar fraction of helium |
| **Argn** | REAL | Molar fraction of argon |
| ► **Cmpr** | REAL | Calculated Compressibility ratio |
| ► **St** | INT | Status register: 0=OK, bits 0-4 indicate errors:<br>bit0=calculation error (see below), bit1=pressure out of range, bit2=temperature out of range, bit4=gas components outside “normal” range. |

The AGA8 Detail calculation is a fairly complicated non-linear calculation that includes a iteration loops (ie. it repeats a calculation many times until the result converges to a solution). The AGA8 function block limits these loops to a maximum of 100 iterations each, to limit the time taken to perform the calculation. A 'calculation error' indicates that after 100 iterations the result was still varying slightly, although a valid result will still be returned by the AGA8 calculation block.

The reason for the variation may be that the input parameters are close to the limits specified in the AGA8 Detailed specification. For further details about the parameter limits and the tolerance levels, please refer to the actual AGA8 Standard. If very precise accuracy is required from the result, the 'calculation error' bit can be used as a warning flag, otherwise it can be ignored.

### 8.16.6 AGA11




Calculates the volumetric flow of gas consistent with AGA Report number 11, *Measurement of Natural Gas by Coriolis Meter*.

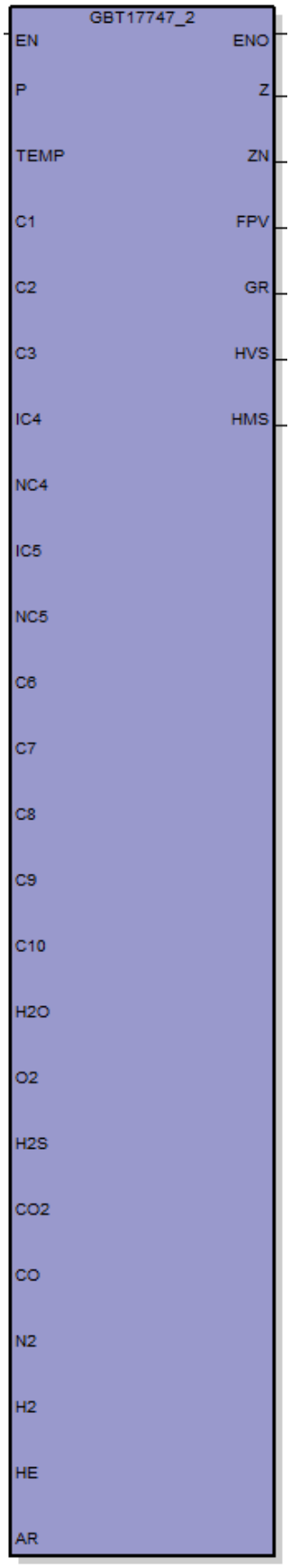
Volumetric flow (QB) is calculated based on mass flow rate (QM) returned from a Coriolis meter and the gas density (PB).

The gas density is calculated from the molar mass ($M_r$) of the gas using the same gas constituent information employed in calculation of the AGA 8 detailed gas compressibility using the formula:

$$p_b = \frac{P_b M_r}{Z_b R T_b}$$

| Parameter | Type | Description |
|---|---|---|
| **PB** | REAL | Gas pressure under contract conditions |
| **TB** | REAL | Gas temperature under contract conditions |
| **QM** | REAL | Mass flow rate from Coriolis meter |
| **Z** | REAL | Gas compressibility under contract conditions<br>This value should be derived from the AGA8_DETAILED function block, using the same molar fractional gas constituents. |
| **METH** | REAL | Molar fraction of methane |
| **N2** | REAL | Molar fraction of nitrogen |
| **CO2** | REAL | Molar fraction of carbon dioxide |
| **ETH** | REAL | Molar fraction of ethane |
| **PROP** | REAL | Molar fraction of propane |
| **H20** | REAL | Molar fraction of water |
| **H2S** | REAL | Molar fraction of hydrogen sulphide |
| **H2** | REAL | Molar fraction of hydrogen |
| **CO** | REAL | Molar fraction of carbon monoxide |
| **O2** | REAL | Molar fraction of oxygen |
| **IBUT** | REAL | Molar fraction of i-butane |
| **NBUT** | REAL | Molar fraction of n-butane |
| **IPEN** | REAL | Molar fraction of i-pentane |
| **NPEN** | REAL | Molar fraction of n-pentatne |
| **HEXA** | REAL | Molar fraction of n-hexane |
| **HEPT** | REAL | Molar fraction of n-heptane |
| **NOCT** | REAL | Molar fraction of n-octane |
| **NONA** | REAL | Molar fraction of n-nonane |
| **DECA** | REAL | Molar fraction of n-decane |
| **HE** | REAL | Molar fraction of helium |
| **AR** | REAL | Molar fraction of argon |
| **► QB** | REAL | Calculated volumetric flow |
| **► D** | REAL | Calculated mass density |
| **► ERR** | BOOL | TRUE if one or more parameters out of range |

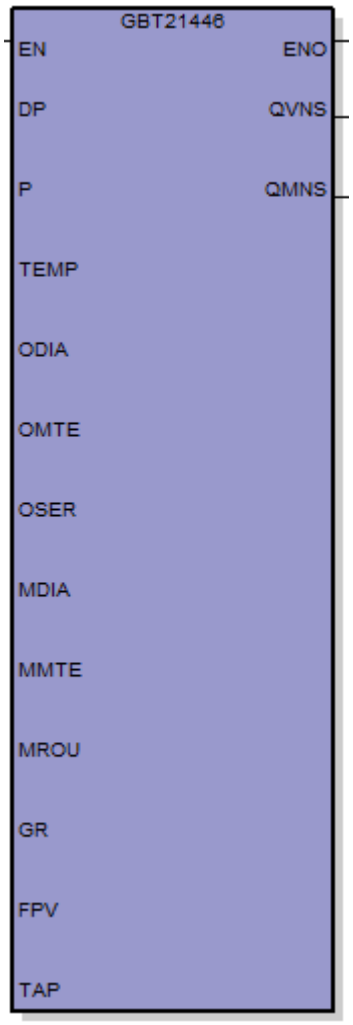
### 8.16.7 GBT17747_2




This function calculates gas compressibility and heating capacity using Chinese gas metering standard GB/T 17747-2.

| Parameter | Type | Description |
|---|---|---|
| **P** | REAL | Gas pressure under flowing conditions (MPa) |
| **TEMP** | REAL | Gas temperature under flowing conditions (°C) |
| **C1** | REAL | Molar fraction of ethane |
| **C2** | REAL | Molar fraction of propane |
| **C3** | REAL | Molar fraction of methane |
| **IC4** | REAL | Molar fraction of i-butane |
| **NC4** | REAL | Molar fraction of n-butane |
| **IC5** | REAL | Molar fraction of i-pentane |
| **NC5** | REAL | Molar fraction of n-pentatne |
| **C6** | REAL | Molar fraction of n-hexane |
| **C7** | REAL | Molar fraction of n-heptane |
| **C8** | REAL | Molar fraction of n-octane |
| **C9** | REAL | Molar fraction of n-nonane |
| **C10** | REAL | Molar fraction of n-decane |
| **H2O** | REAL | Molar fraction of water |
| **O2** | REAL | Molar fraction of oxygen |
| **H2S** | REAL | Molar fraction of hydrogen sulphide |
| **CO2** | REAL | Molar fraction of carbon dioxide |
| **CO** | REAL | Molar fraction of carbon monoxide |
| **N2** | REAL | Molar fraction of nitrogen |
| **H2** | REAL | Molar fraction of hydrogen |
| **HE** | REAL | Molar fraction of helium |
| **AR** | REAL | Molar fraction of argon |
| **► Z** | REAL | Calculated standard compressibility under flow conditions |
| **► ZN** | REAL | Calculated supercompressibility under reference conditions |
| **► FPV** | REAL | Calculated supercompressibility under flow conditions |
| **► GR** | REAL | Calculated relative density |
| **► HVS** | REAL | Calculated volume heating capacity |
| **► HMS** | REAL | Calculated mass heating capacity |

### 8.16.8 GBT21446




This function blocks calculates natural gas volume flow rate based on standard orifice metering. This calculation is described in the Chinese standard GB/T 21446.

| Parameter | Type | Description |
|---|---|---|
| **DP** | REAL | The differential pressure returned from the orifice meter (Pa) |
| **P** | REAL | Absolute static gas pressure under flowing conditions (MPa) |
| **TEMP** | REAL | Temperature under flowing conditions (°C) |
| **ODIA** | REAL | Orifice plate bore diameter (m) |
| **OMTE** | REAL | Orifice plate thermal expansion coefficient |
| **OSER** | REAL | Service time |
| **MDIA** | REAL | Meter tube internal diameter (m) |
| **MMTE** | REAL | Meter tube thermal expansion coefficient |
| **MROU** | REAL | Absolute roughness |
| **GR** | REAL | Specific gravity |
| **FPV** | REAL | Super compressibility under flow conditions |
| **TAP** | REAL | Mode of pressure tap |
| ► **QVMS** | REAL | Calculated volume flow rate |
| ► **QMNS** | REAL | Calculated mass flow rate |

## 8.17 Obsolete Function Blocks

The following function blocks currently appear in the ISaGRAF selection list, but are obsolete or not supported. These function blocks should not be used.

### 8.17.1 AGA1

This function block is not supported.

### 8.17.2 AGA9

The AGA9 report concerns the measurement of the flow of natural gas using an Ultrasonic meter. This report refers the reader to the calculations in the AGA7 report, hence the AGA7 function block should be used for these applications.

### 8.17.3 DNPM_ANALOG_COMMAND

This has been superseded by the DNPM_ANALOG_16BIT_COMMAND, DNPM_ANALOG_32BIT_COMMAND and DNPM_ANALOG_FLOAT_COMMAND function blocks.

### 8.17.4 IMAGE

This function block is not supported.

### 8.17.5 TRIO

This function block is not supported.

# 9. ISaGRAF - Logic Examples

A Toolbox PLUS+ project containing all the Ladder Diagram logic examples below is available from the support website.

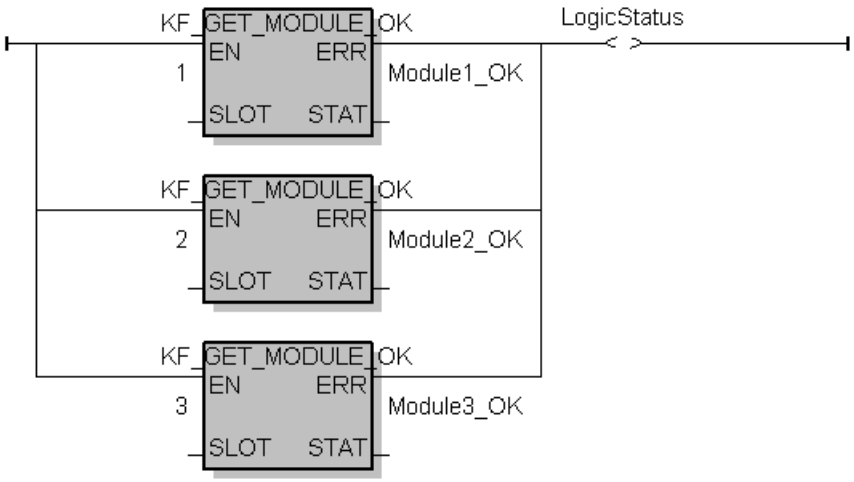
| Example | Description |
|---|---|
| Detecting Modules | How to check that each module is present on the backplane |
| Scaling | How to convert an analog input into engineering units |
| Hours On | The total number of hours that a variable has been active |
| Counting Pulses and Starts | The total number of pulses or starts (OFF to ON transitions) of a variable |
| Flow Totalisation | Counting volume from a flow-rate |
| Daily Totals | Rolling over totals at midnight to create yesterday and current totals |
| Polling | How to poll a Kingfisher RTU |
| Exception Reporting Digitals | How to send a message when a digital variable changes state |
| Exception Reporting Analog variables | How to send a message when an analog variable changes significantly |
| Event Logging | How to create periodic or change initiated event logs |
| Modbus Protocol | How to configure and use the Modbus protocol for Master and Slave |
| Modbus Protocol: Extended Addressing | Configuring Modbus communications using addresses larger than 255 (outside the Modbus address space) |
| Allen Bradley Protocol | How to configure and use the Allen Bradley DF1 protocol |

## 9.1 Detecting Modules

The logic below checks that the modules detected in slots 1, 2 and 3 match the configuration loaded in the RTU (from Toolbox PLUS). The variables **Module1_OK**, **Module2_OK**, **Module3_OK**, **ModulesOK** and **LogicStatus** are all BOOL type variables. Please see the topic ISaGRAF – Custom Function Blocks for function block details.

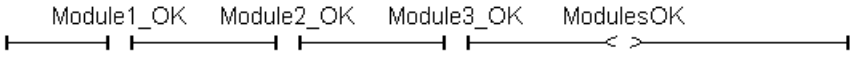
(* Check modules in slots 1, 2 and 3 match the configuration *)

[1]




(* Check that all modules are OK *)

[2]



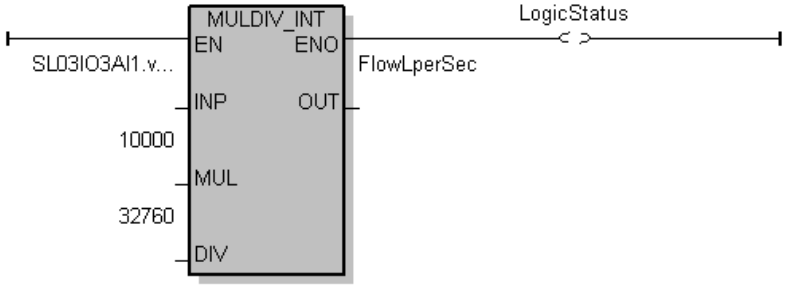
## 9.2 Scaling

The logic below converts an analog input (SL03IO3AI1.value) into engineering units. The input has a raw range of 0-32760. The input is divided by 32760 and then multiplied by 10000 to convert it to a range of 0-10000 (this value can then be displayed as 0-1000.0 or 0-100.00 L/s in SCADA software). **FlowLperSec** is a DINT variable and **LogicStatus** is a BOOL variable.

Please see the topic ISaGRAF – Custom Function Blocks → Maths → Multiply Divide Integer for function block details.

(* Convert Analog Input 1 to Engineering Units 0-10000 = 0-100.00 L/s *)

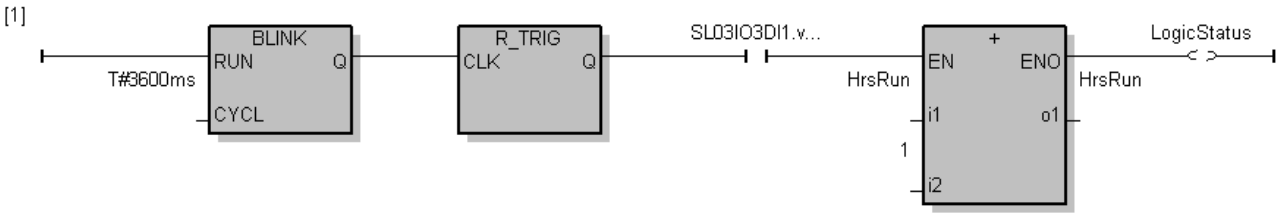
[1]



## 9.3 Hours ON

Every 3600 milliseconds (3.6 seconds or 1/1000th of an hour) the logic checks if **SL03IO3DI1.value** is ON and if it is, **HrsRun** (a DINT variable) is incremented. **HrsRun** then contains the number of 0.001 hour intervals that the input is ON i.e. 0 to 999,999 = 0 to 999.999 Hrs. **HrsRun** can be rolled over at midnight to obtain daily totals.

```
(* Every 3600 ms (3.6 seconds) check if DI1 is ON. If ON, increment HrsRun *)
```



## 9.4 Counting Pulses And Starts

Low speed pulses (up to 5 pulses per second [Hz]) and starts are both counted the same way. The logic checks for an OFF to ON (or ON to OFF) transition of a digital input and increments a counter when a transition is detected.

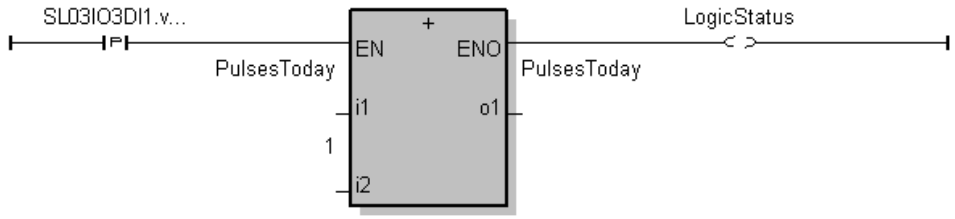
The actual pulse rate that the RTU can count depends on how fast the logic and IO are processed. Since pulses are counted by counting the rising or falling edges of digital inputs, the logic and IO needs to be processed fast enough to allow the RTU to register the pulse in the ON and the OFF states.

The default logic (and IO) cycle time is 100 ms and is configured in ISaGRAF from **Project, Run Time Settings**, **Settings**, **Cycle Timing**. The minimum cycle time that can be used will vary according to the number and type of modules in the RTU.

To count higher pulse rates (up to 10 kHz), a DI-10 or DI-5 digital input module can be used. These modules have dedicated hardware counters and do not rely on the logic cycle time to count pulses.

The logic below shows how to count pulses (or starts) using a positive (rising) edge trigger. Every time there is a new pulse (from **SL03IO3DI1.value**), the **PulsesToday** variable (a DINT variable) is incremented**. LogicStatus** is a BOOL variable.

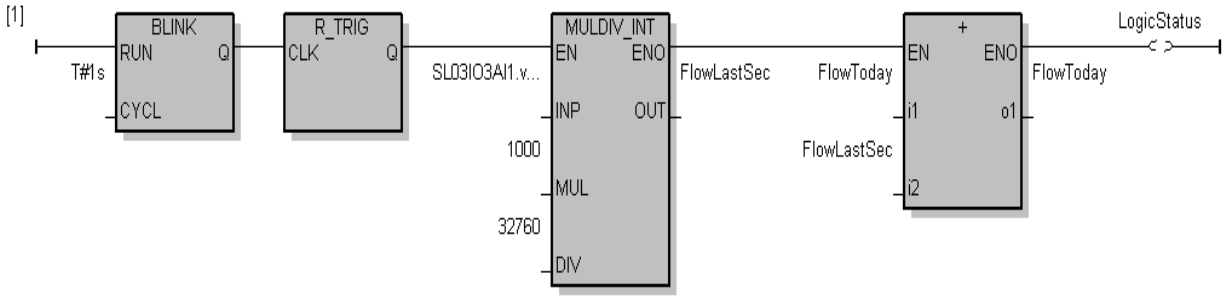
```
(* Count Digital Input 1 pulses (positive edge trigger) *)
[1]
```



## 9.5 Flow Totalisation

The following example shows how to accumulate a flow volume from a flow-rate analog input (**SL03IO3AI1.value**). For this example, the flow-rate engineering units are 4-20mA=0-1000 L/s. Each second, the number of litres that have flowed (**FlowLastSec** [a DINT variable]) is calculated by dividing the analog input by 32760 (the raw analog input range) and then multiplying by 1000 (the high limit of the engineering units). This number of litres is then added to the **FlowToday** total (a DINT variable).

(* Convert Analog Input 1 (0-1000 L/s) to number of litres each second. Add to flow total. *)



## 9.6 Daily Totals

Daily totals are created by rolling over current totals at midnight. The current totals are copied to yesterday totals and then the current totals are reset.

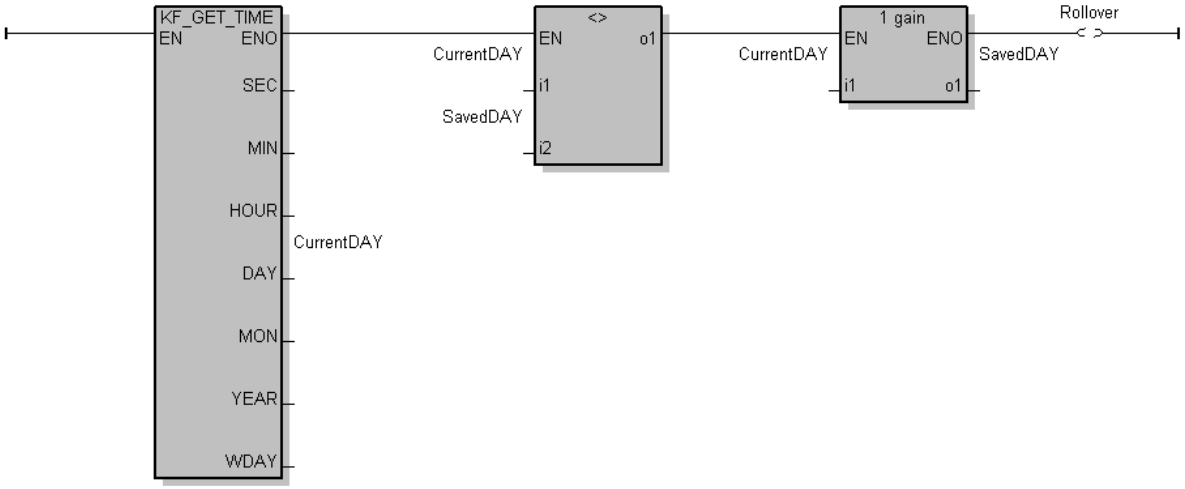
In the example below, **CurrentDAY** and **SavedDAY** are DINT variables and **Rollover** is a BOOL variable. When the value for **CurrentDAY** (of the month) changes at midnight, **SavedDAY** is updated and **Rollover** is set TRUE for one logic scan.

When **Rollover** is TRUE, the **PulsesToday** total is copied to **PulsesYesterday** and then zero is copied to **PulsesToday**.

Please see the topic ISaGRAF – Custom Function Blocks → RTU System Data → Get Time for function block details.

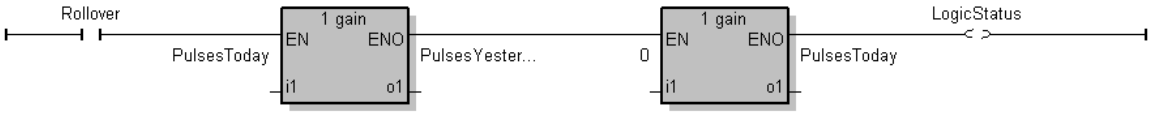
(* Check current day of month. If day has changed, trigger a rollover. *)

[2]




(* Rollover Pulse Total *)

[3]



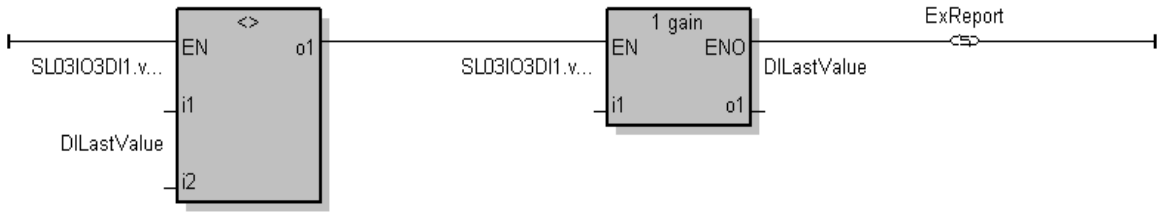
## 9.7 Exception Reporting Digitals

An exception report can be generated when a single digital bit changes state or when any of the bits in a variable change state.

### 9.7.1 Monitoring A Single Bit

In the example below, a single digital input variable from an IO-3 module (**SL03103DI1.value**) is monitored for change. If the input changes, an exception report flag is set (**ExReport**). **DILastValue** and **ExReport** are BOOL variables.

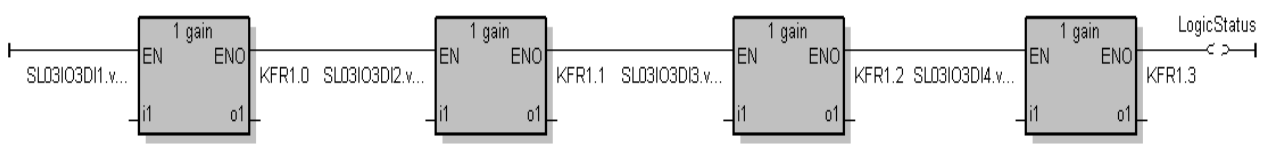
```
(* Set ExReport flag if Digital Input 1 changes *)
[1]
```



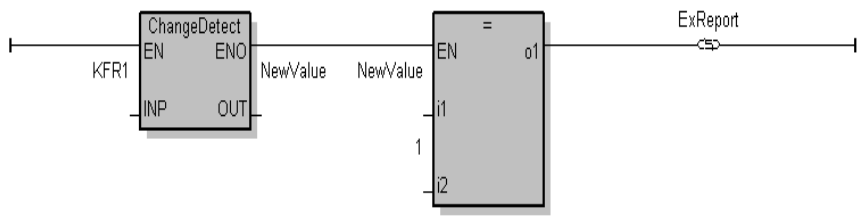

### 9.7.2 Monitoring Multiple Bits

In the example below, the four digital inputs from an IO-3 module (**SL03IO3DI1.value**, **SL03IO3DI2.value**, **SL03IO3DI3.value** and **SL03IO3DI4.value**) are copied to Kingfisher Register Variable 1 (**KFR1**). If Register Variable 1 changes (ie. if any bit changes), an exception report flag is set. **NewValue** is a DINT variable. **LogicStatus** and **ExReport** are BOOL variables.

```
(* Copy Digital Inputs to Kingfisher Register Varable 1 *)
[2]
```




```
(* Set ExReport flag if Kingfisher Register Variable 1 bits have changed *)
[3]
```



### 9.7.3 Sending The Exception Report

Create one or more Kingfisher Local Register variables to send to the remote RTU. Example: **KFR1** (of type DINT).

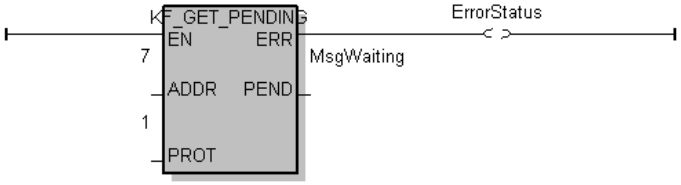
Create an INT_ARRAY variable to be used for the **REG** parameter (example: **LocalRegisters**). Configure the **Initial Value** parameter to correspond to the Local Register Variables to send to the remote RTU. Example: to send Local Register Variables 1 to 5, configure an Initial Value of **1,2,3,4,5** (please see below for details)

Configure a **KF_TX_DATA** function block in an ISaGRAF logic program, using the appropriate variables (ie. correct data type) or constants for the inputs and output. Assign the INT_ARRAY variable created above to the **REG** input parameter.

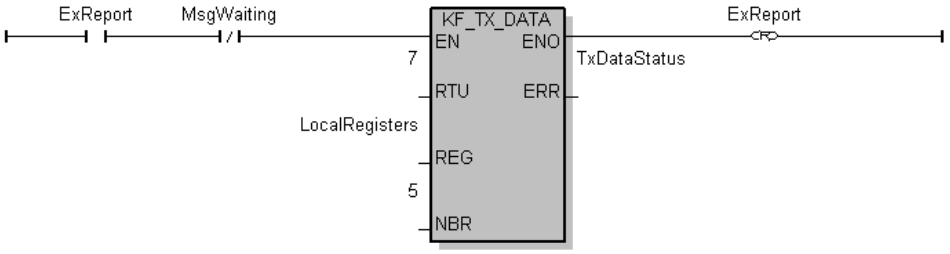
- When the function is executed, the RTU will get values from the specified local register variables and send them to the remote RTU.
- In the remote RTU the data will appear in Kingfisher network register variables (for CP-30/G30 RTUs) or network registers (for CP-11, PC-1 RTUs).

The logic below sends a message to RTU7 when the port is free and the **ExReport** flag (from above examples) has been set. **MsgWaiting** and **ErrorStatus** are BOOL variables. **TxDataStatus** is a DINT variable.

```
(* Check if a message for RTU7 is waiting for a reply *)
[4]
```




```
(* Send Exception Report if new data and not waiting for a reply to a previous message *)
[5]
```



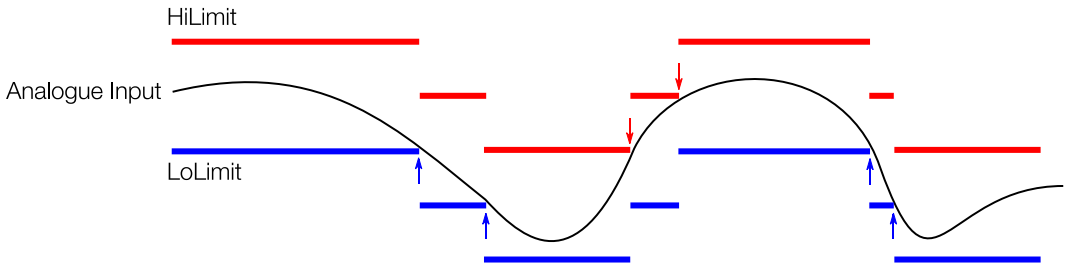

The **LocalRegisters** variable used above is defined as follows:

| | Name | Type | Initial Value |
|---|---|---|---|
| | LocalRegisters | INT_ARRAY | 1,2,3,4,5 |

The initial value of **LocalRegisters** allows the KFR1 to KFR5 variables to be transmitted to the remote RTU.

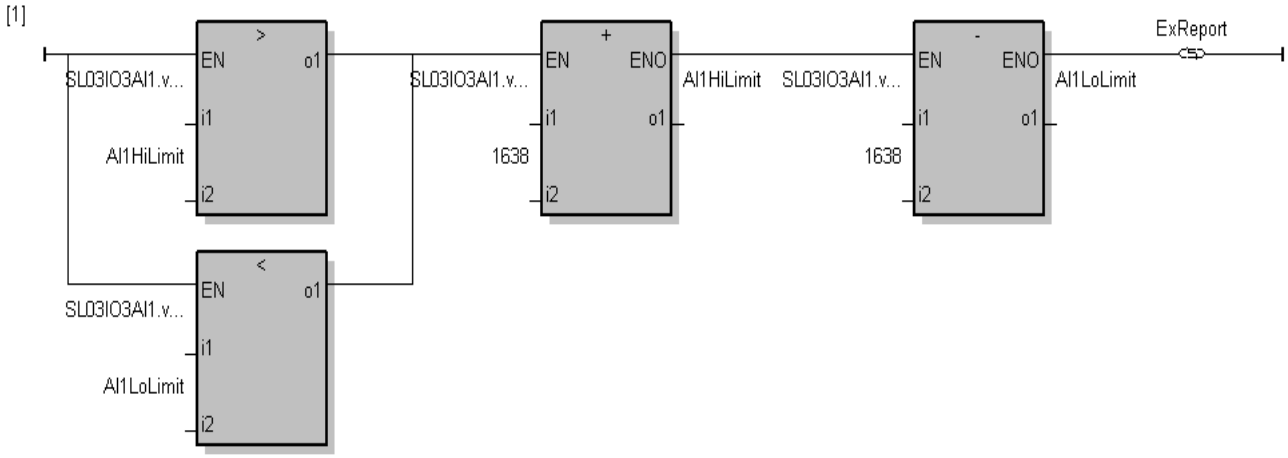
## 9.8 Exception Reporting Analog Variables

The example below shows how to trigger an exception report when an analog input (**SL03IO3AI1.value**) changes by 5% (of the analog range) from the last reported value. This is done by using two DINT variables (**AI1HiLimit**, **AI1LoLimit**) and a constant. The variables are used to store the last reported value plus the constant and the last reported value minus the constant. When the analog value moves above or below these values, an exception report is triggered and the thresholds are updated (as illustrated below).

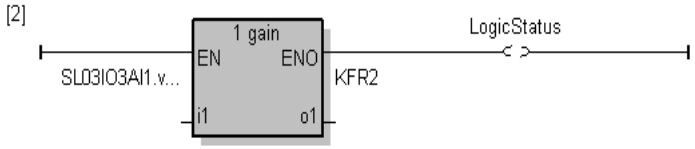



The constant to use is calculated as percentage of the analog or register range. For analog inputs which have a range of 0-32760 (32767 for an AI-10), a 1% change is represented in the RTU by a change of about 328 (0.01 x 32760). Similarly, a 5% change is represented in the RTU by a change of 1638 (0.05 x 32760).

(* Set ExReport Flag if Analog Input 1 changes by 5% (1638) *)




(* Copy Analog Input 1 to Kingfisher Register Variable 2 *)




To send the new analog value (that was copied to the **KFR2** in the logic above) to another RTU, please see the topic above, Logic Examples – Exception Reporting Digitals.

## 9.9 Event Logging

The RTU is able to keep a record of variables and their values over time. Kingfisher or DNP3 variables can be logged on change or periodically. The maximum number of event logs that an RTU will keep is configured in the **RTU Properties**, **General** tab. By default, the RTU will keep up to 10,000 event logs.

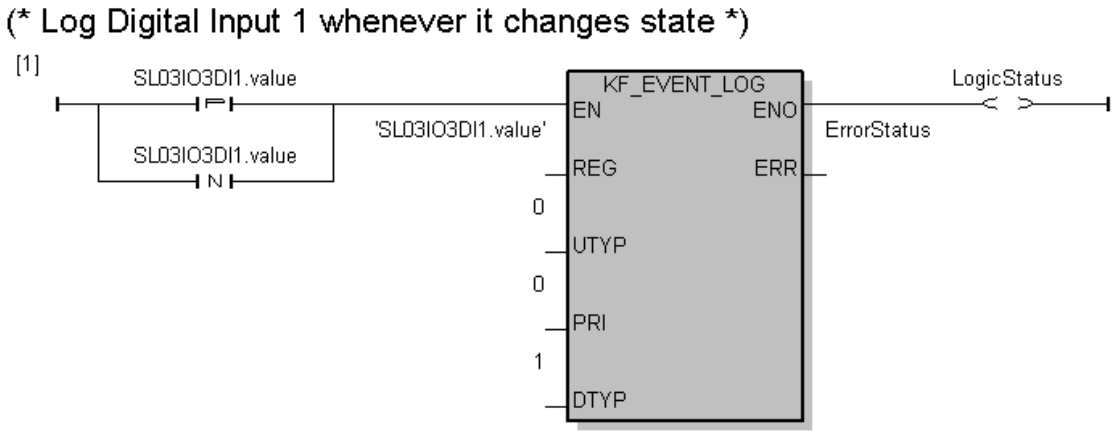
For the two examples below, **ErrorStatus** is a DINT variable and **LogicStatus** is a BOOL variable.

### 9.9.1 Logging Kingfisher Variables

The example below creates an event log whenever digital input 1 (**SL03IO3DI1.value**) changes state.

**REG** is configured with the string **'SL03IO3DI1.value'** (including the single quotation marks).

Please see the topic ISaGRAF – Custom Function Blocks, Event Logging, Event Log for function block details.

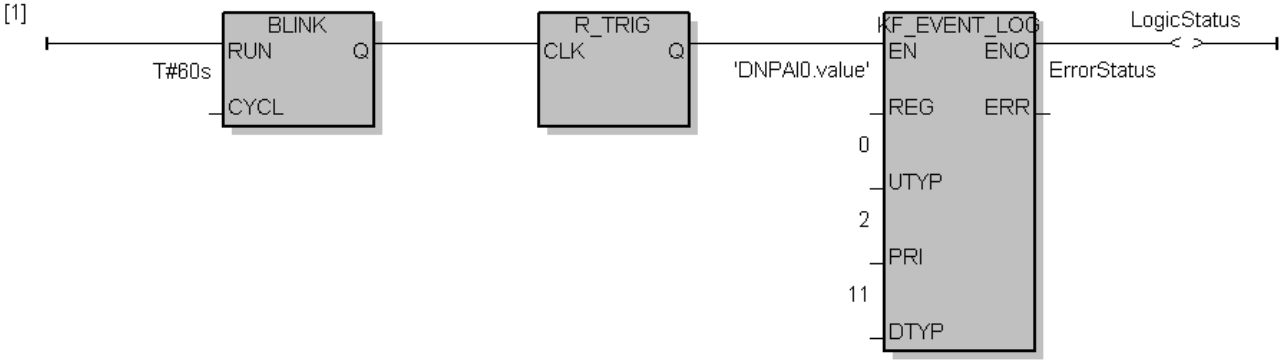


### 9.9.2 Logging DNP Variables

The example below creates periodic event logs (every 60 seconds) for DNP Analog Input 1 (**DNPAI0**) with a priority of 2 (class 2 data).

Please see the topic ISaGRAF – Custom Function Blocks, Event Logging**, Event Log** for function block details.

(* Log DNP Analog Input 1 every 60 seconds *)



## 9.10 Logic Examples – Polling

Polling is usually performed by the master RTU in order to get a regular update of remote RTU data and to determine if communications to the remote RTUs have failed.

Kingfisher protocol allows for full-duplex communications which means the RTU can simultaneously transmit and receive. However, since most radios are half-duplex, which means that the RTU cannot transmit and receive simultaneously, it is necessary to force the RTU to wait for a reply to each transmit message. Unless the RTU is forced to wait, it will transmit all the polling messages one after the other - which can take a few seconds. Because of the delay, the first polling message will have timed out before the RTU is able to receive a reply as the RTU is still busy transmitting. The following examples show how to force the RTU to wait for a reply to each message when polling.

Create one or more Kingfisher Network Register Variables to store the data received from the remote RTU. Example: **KF7R1, KF7R2, KF7R100**.

Create an INT_ARRAY variable to be used for the **REG** parameter (example: **RTU7Registers**). Configure the **Initial Value** parameter to correspond to the Local Register Variables to poll from the remote RTU. Example: to poll Local Register Variables 1, 2 and 100 use an Initial Value of **1,2,100** (please see below for details)

Add the **KF_RX_DATA** function to an ISaGRAF logic program and configure each of the input parameters. Assign the INT_ARRAY variable created above to the **REG** input parameter.

- When the function is executed, the RTU will get the local register variables from the remote RTU and store them in the corresponding network register variables in the local RTU.

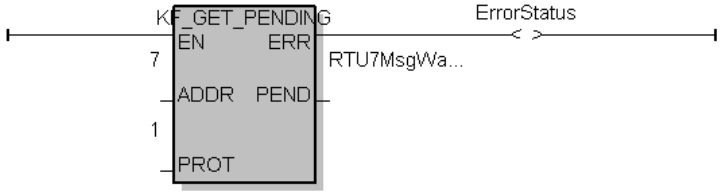
Please see the topic ISaGRAF – Custom Function Blocks, Kingfisher Messages, Rx Data for function block details.

### 9.10.1 Basic Polling

The logic below polls RTU7 Local Register Variables 1, 2 and 100 every 60 seconds when the port is free. **RTU7MsgWaiting**, **ErrorStatus** and **PollRTU7** are BOOL variables and **RTU7Registers** is an INT_ARRAY variable (as detailed below).

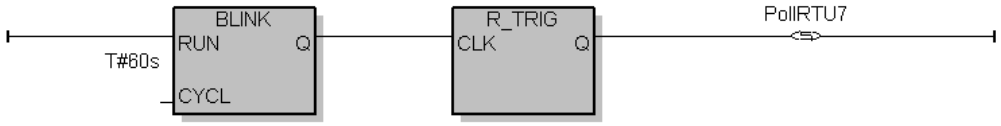
(* Check if a message for RTU7 is waiting for a reply *)

[1]



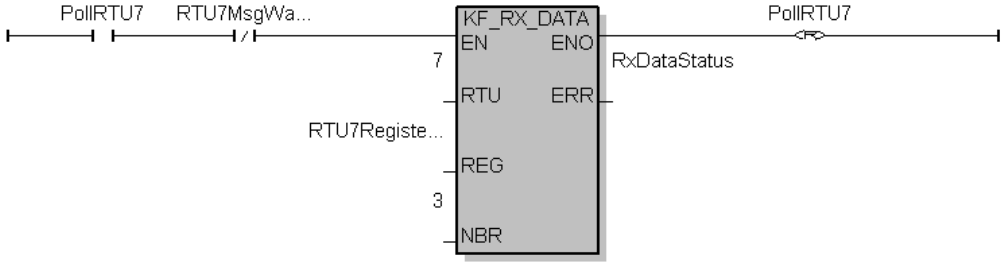

(* Set RTU7 poll flag every 60 seconds *)

[2]




(* Poll RTU7 if poll flag set and not waiting for a reply to a previous message *)

[3]




The **RTU7Registers** variable used above is defined as follows:

| | Name | Type | Description | Initial Value |
|---|---|---|---|---|
| | RTU7Registers | INT_ARRAY | | 1,2,100 |

The initial value of **RTU7Registers** allows the **KFR1**, **KFR2** and **KFR100** variables to be polled from the remote RTU.

### 9.10.2 Polling after Data Has Expired

If a remote RTU has exception reported to the master RTU recently, it is not necessary to poll the remote RTU until the data is older than X minutes (where X is ideally a SCADA set-point with a default value of say 10 minutes). If exception reports are generated frequently, it may never be necessary to poll the remote RTU. Communication statistics are still accumulated as a success is recorded for each exception report received. Communication Fails are also recorded as the master will still check communications every X minutes if it has not received a message from the remote RTU during that time. Note: the variables that are exception reported should be the same as the variables that are polled.

**Advantages:**

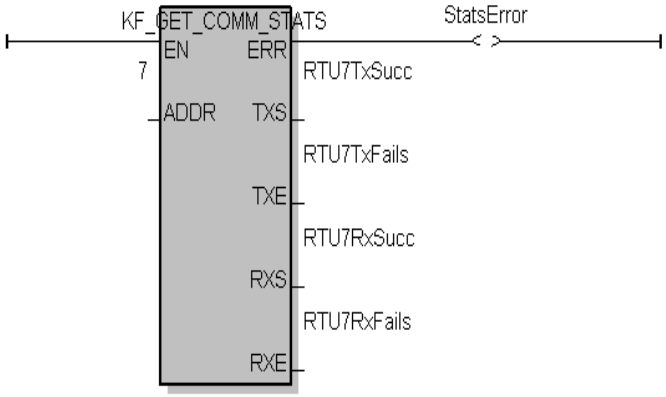
- Minimises communications over the network
- User can set the maximum age of data before a poll

**Disadvantages:**

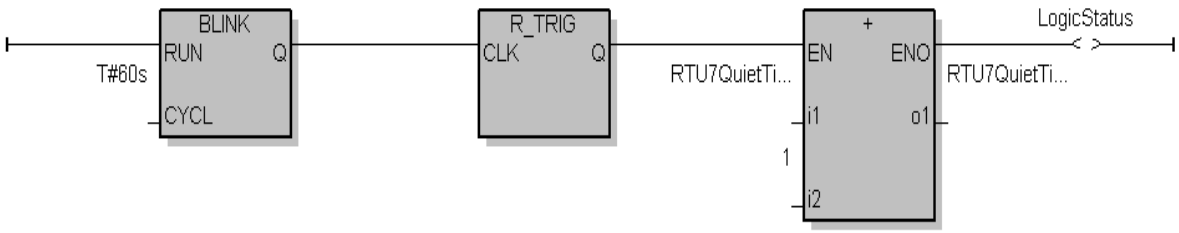
- Takes more effort to test

The logic below polls RTU7 Local Register Variables 1, 2 and 100 when the data is too old. The maximum age of data (in minutes) is set in the variable **MaxDataAge** (of type DINT). If a message is received from RTU7 or a poll has occurred, the **RTU7QuietTimer** (of type DINT) is reset to 0. **StatsError**, **LogicStatus**, and **RTU7MsgWaiting** are BOOL variables. **LastRxSucc**, **RTU7TxSucc**, **RTU7TxFails**, **RTU7RxSucc** and **RTU7RxFails** are all UDINT variables.

```
(* Check comms statistics for RTU7 *)
[1]
```

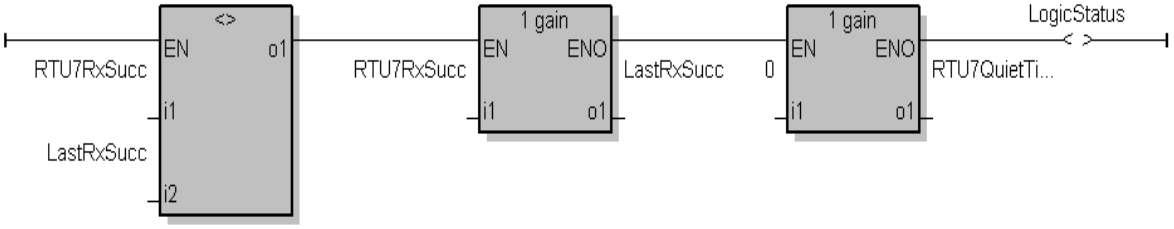



```
(* Increment RTU7 Quiet Timer each Minute *)
[2]
```



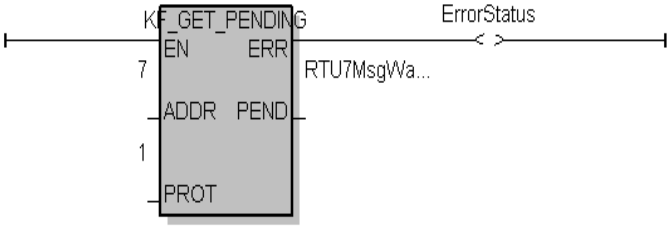
(* Reset RTU7 Quiet Timer if a new message is received from RTU7 *)

[3]



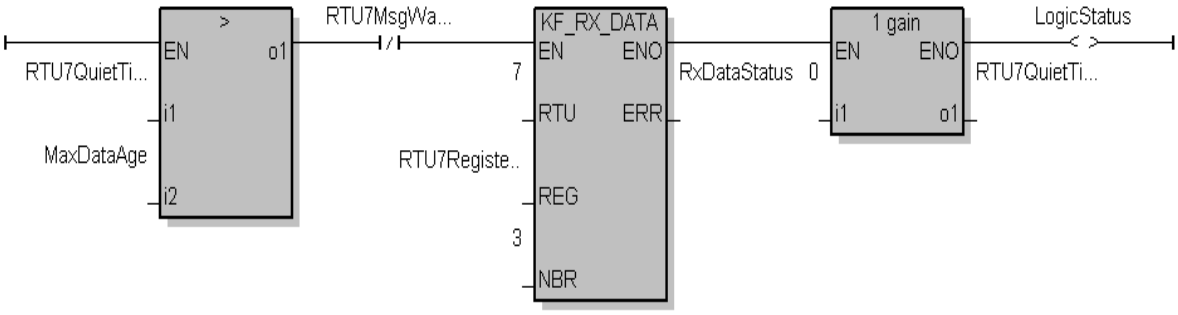

(* Check if a message for RTU7 is waiting for a reply *)

[4]




(* Poll RTU7 if no message received for more than 'MaxDataAge' minutes *)

[5]




The **RTU7Registers** variables used above is defined as follows:

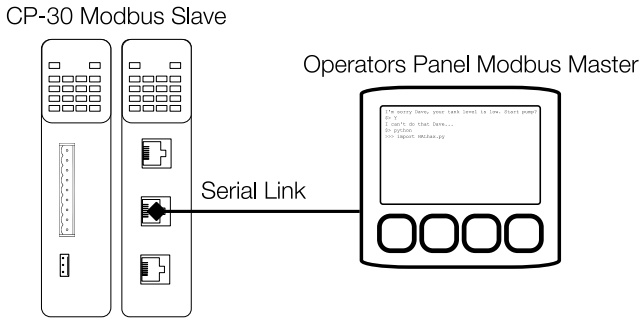
| | Name | Type | Description | Initial Value |
|---|---|---|---|---|
| | RTU7Registers | INT_ARRAY | | 1,2,100 |

The initial value of **RTU7Registers** allows the **KFR1**, **KFR2** and **KFR100** variables to be polled from the remote RTU.

## 9.11 Modbus Protocol

Kingfisher RTUs can be configured to behave as a Modbus master or slave.
The following examples outline both configurations.

### 9.11.1 Setting up an RTU to be a Modbus Slave



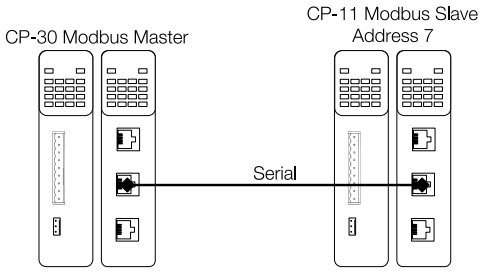

Add the **Modbus RTU** protocol to the RTU (RTU Configuration - Protocols). Alternatively **Modbus TCP** can be used an Ethernet port.

- Add serial port 2 to the RTU configuration (RTU Configuration - Ports)

Edit serial port 2 (RTU Configuration - Ports). From the Settings tab of port 2, set **Bits per second** to 9600 and enable the **Modbus RTU** protocol.

- Create local Modbus variables to be polled by the Modbus Master device (Dictionary - New Variables). Example: MODH71 - Modbus Holding register 71 (40,071). Please see the appendix RTU Data - Protocols, Modbus Data for Modbus variable formats and addressing details.
- Add an ISaGRAF logic program to copy data into the Modbus variables as required (RTU Configuration - Programs). Note: an Initial Value can be configured and an IO channel can be mapped to each Modbus variable.
- Download the Configuration and Logic (even if not using a logic program) to the RTU.

### 9.11.2 Setting up an RTU to be a Modbus Master




Add the **Modbus RTU** protocol to the RTU (RTU Configuration - Protocols). Alternatively **Modbus TCP** can be used on an Ethernet port.

- Add serial port 2 to the RTU configuration (RTU Configuration - Ports).

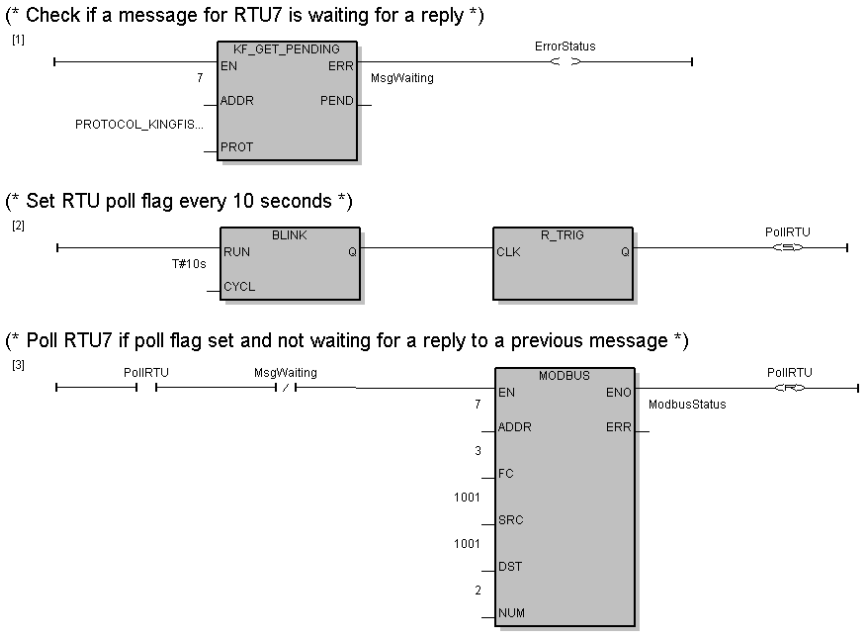
Edit serial port 2 (RTU Configuration - Ports); set **Bits per second** to 9600 and enable the **Modbus RTU** protocol.

Create Modbus Variables to store the data polled from the Modbus Slave device (Dictionary - New Variables). Example: **MOD7H1001** to **MOD7H1005** (Modbus device 7, Holding Registers 1001 to 1005). Note: Local Registers (#Rx) polled from a PC-1, LP-x or CP-11/21 RTU start at 1001. Therefore to poll #R10, Modbus holding register 1010 would be polled.

Create local Modbus variables that correspond to the registers to be *written* to the Modbus Slave device. Example: **MODH1014** to **MODH1016** (Modbus Holding registers 1014 to 1016). Please see the appendix RTU Data - Protocols, Modbus Data for Modbus variable formats and addressing details. To read or write floating point values using the Modbus protocol, please see the FPACK and FUNPACK function blocks.

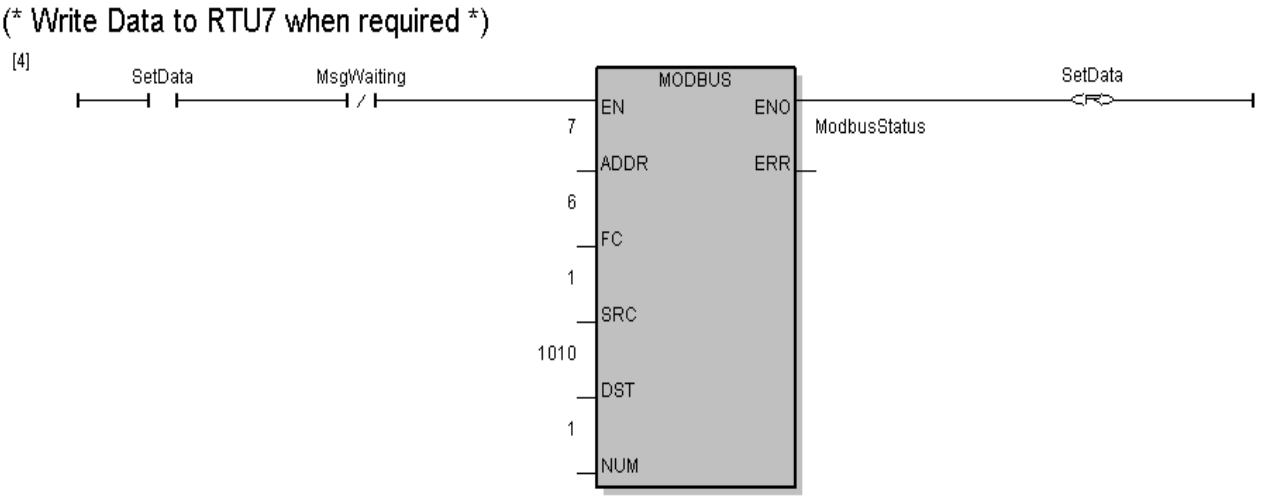
- Configure a Modbus function block in an ISaGRAF logic program, using the appropriate variables or constants for the inputs and output. An example is shown below.
- Download the Configuration and Logic to the RTU.

The logic below polls Local Registers R1 and R2 (Modbus holding registers 1001 and 1002 respectively) from CP-11 RTU7 using the Modbus protocol every 10 seconds. The data is stored in MOD7H1001 and MOD7H1002. ErrorStatus, MsgWaiting and PollRTU are BOOL variables. **ModbusStatus** is a DINT variable.




**Note 1:** by changing the **DST** setting above to 1, Local Registers **R1** and **R2** from **RTU7** could be stored in **MOD7H1** and **MOD7H2** instead of **MOD7H1001** and **MOD7H1002**.

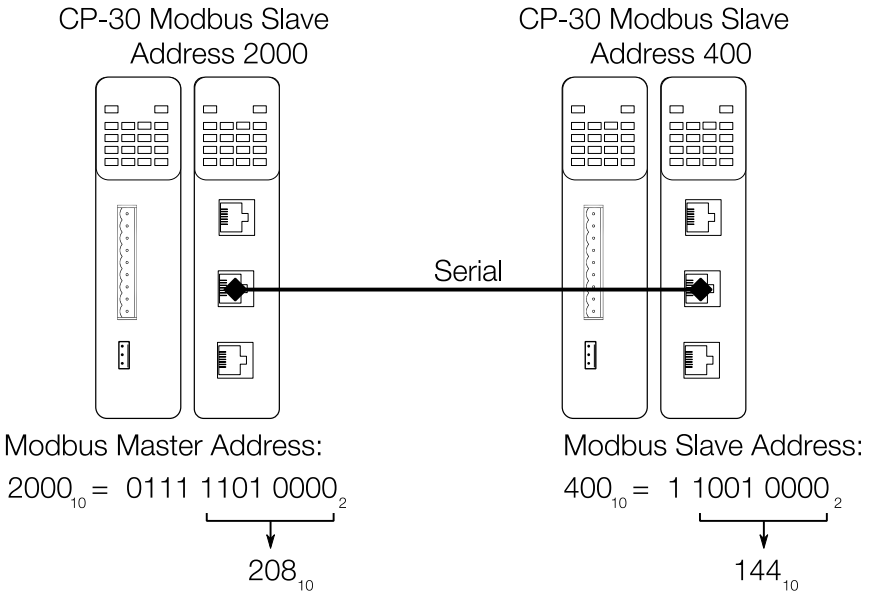
**Note 2:** When writing data to a PC-1, LP-x or CP-11/21 RTU using the Modbus protocol, data is stored in Local Registers of the destination PC-1, LP-x or CP-11/21 RTU. The logic below writes the value from the local MODH1 variable to **#R10** [Modbus address 41010] in CP-11 RTU7.



## 9.12 Modbus Extended Addressing

The CP-30 and G30 processors are capable of addressing from 0 to 65520, while Modbus is limited to 0 to 255; special consideration needs to be taken when integrating a CP-30/G30 RTU into a Modbus network.

If the RTU address exceeds 255, only the lower 8 bits are preserved in Modbus and become the address, for example:




This means that CP-30 or G30 with address 512 will appear as a Modbus device with address 1.

For further clarification please refer to the Kingfisher Examples. The project **Modbus Slave Extended Addressing / Modbus Master Extended Addressing** will behave identically to their standard counterparts **Modbus Slave / Modbus Master**.

## 9.13 Allen Bradley Protocol

### 9.13.1 Communicating With An Allen Bradley SLC500 PLC

- Add the Allen Bradley DF1 protocol to the RTU (RTU Configuration - Protocols).
- Add a serial port to the RTU configuration (RTU Configuration - Ports).
- Edit the serial port (RTU Configuration - Ports). From the Settings tab of the port, set **Bits per second** to the desired baudrate and enable the Allen Bradley protocol.
- Add the station address of the PLC to the RTU's Route list (RTU Configuration - Routes). Note: the station address cannot already be used by another RTU in the list.
- Create one or more Kingfisher Local Register variables to send to the PLC or to store data from the PLC. Example: **KFR1** (of type DINT) to **KFR200**.
- Configure an Allen Bradley function block in an ISaGRAF logic program, using the appropriate variables (ie. correct data type) or constants for the inputs and output. An example is shown below.
- Download the Configuration and Logic to the RTU.

The logic below polls 50 registers starting at N10:1 from an Allen Bradley PLC every 10 seconds and stores the data in KFR10 to KFR60. **ErrorStatus**, **MsgWaiting** and **PollNow** are BOOL variables. **ABErrorStatus** is a BYTE variable.

```
(* Check if an Allen Bradley message for address 26 is waiting for a reply *)
[1]
                    KF_GET_PENDING
                    EN          ERR  ErrorStatus
                26  ADDR        PEND MsgWaiting
                18  PROT

(* Set poll flag every 10 seconds *)
[2]
                    BLINK                R_TRIG
                    RUN         Q        CLK         Q  PollNow
                T#10s
                    CYCL

(* Poll Allen Bradley PLC if poll flag set and not waiting for a reply to a previous message *)
[3]
PollNow  MsgWaiting                ABDF1_RX
                                   EN          ENO  PollNow
                                26 ADDR        ERR  ABErrorStatus
                                 0 PLC
                                50 NUM
                           'N10:1' SRC
                                10 DEST
```

**Note 1:** The simplest way to connect between an RTU serial port and an Allen Bradley PLC is to use an RS232 null modem cable (can use the Semaphore ADP-05 adaptor and an

RJ45 to RJ45 patch lead). An Allen Bradley SLC5/03 CPU has a DB9 male port while the SLC5/02 CPU has an RS485 RJ45 port and may need to have a communications module installed for RS232 (RS485 can be used instead).

**Note 2:** If a 1785-KE interface module is used between the PLC and the RTU, the 1785-KE station number must also be configured in the Route list. The PLC should then be configured as an indirect link via the 1785-KE station address.

# 10. Redundancy

Redundancy allows RTUs installed in critical applications to continue operating normally when a power supply, a processor or a communications link fails. Each RTU can also be monitored and controlled by two or more PCs running SCADA software as detailed in the topic Redundant PCs.

## 10.1 Redundant Processors

### 10.1.1 Overview

To configure a redundant processor, simply add a second CP30 module to the RTU. The second CP30 will act as a backup and will automatically take over in the event that the primary processor fails. (Note that redundant processor operation is not available on G30 RTUs.)

One CP30 processor must be installed in an even-numbered slot of the RTU backplane and one processor must be installed in an odd-numbered slot of the backplane. The processor installed in the even-numbered slot is called the **primary** processor, while the processor installed in the odd-numbered slot is the **secondary** processor.

Initially, the primary processor operates in **active** mode: it scans the I/O, runs the logic, and initiates and responds to communication messages. The secondary processor remains in **standby** mode ready to switch to active mode if the primary processor fails. The standby processor still responds to messages but it does not run its logic or initiate messages.

To keep the standby processor up to date, logic information, event logs and communication indices are regularly updated in the standby processor while the active processor is running.

The **F3** LED indicates active or standby mode of each processor. When steady, the processor is in active mode. When flashing (1 Hz), the processor is in standby mode.

The processor modules do not need to be installed physically next to each other. They just need to be in odd and even-numbered slots somewhere on the RTU backplane.

### 10.1.2 Active and Standby Processor Operation

In a single processor system, the CP30 is responsible for scanning input modules, processing logic, updating output modules and initiating and responding to communications messages. In a redundant processor system, the active processor still does all this, but also regularly updates the standby processor with the following items:

- ISaGRAF symbols (dictionary variables) that were modified during the last cycle of logic execution. This includes function block parameters.
- New Kingfisher and DNP3 event logs. The values, flags and time stamps as recorded in the active processor module are all copied. Note: event logs recorded before the last power cycle are not copied to the standby processor. If all the event logs are cleared in the active processor (e.g. using logic), the event logs in the standby processor will also be cleared.
- DNP3 event list indices

- Time is synchronised every second. Note: protocols are only enabled on the standby processor after receiving the first time synchronisation from the active processor. This ensures that I/O points have been read from the hardware modules before the protocol is enabled.

The standby processor's responsibilities include:

- Receiving the above data from the active processor and updating its local copy of variable values and event logs.
- Responding to incoming communications read requests. Requests to write data to the RTU will be rejected, as will requests to read DNP3 events. DNP3 static values (class 0 data) will be returned.
- Monitoring backplane activity. If no activity is seen for 500ms then it is concluded that the primary processor has failed. The standby processor will then become the active processor.

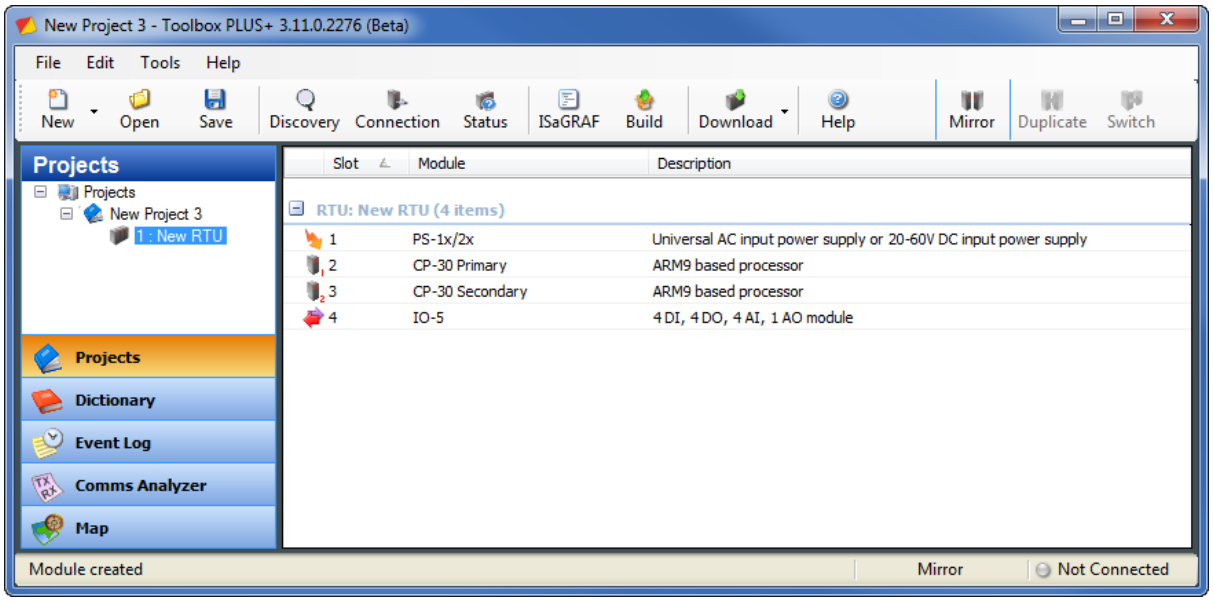
When a CP30 starts up, it first monitors backplane activity for a period of time. It will only decide to become the active processor if no activity is detected. This initial timeout is 500ms for the primary (even slot) processor, and 10,000ms for the secondary. This asymmetric setting ensures that when both are powered up simultaneously, it will be the primary processor that initially becomes active.

Note that apart from this bias on initial startup, the primary and secondary processors are otherwise treated equally. For example, if the primary CP30 is removed (causing the secondary to become active) and then replaced, the secondary will continue to operate as the active processor. The primary will only become active in the event that the secondary fails.

### 10.1.3 Configuring Redundant Processors

Ensure one processor is installed in an even slot on the backplane and the other processor is installed in an odd slot.

In Toolbox PLUS+, after creating a new CP-30 RTU configuration, add a second CP-30 module to the RTU. Toolbox PLUS+ will automatically label the processors "Primary" and "Secondary".



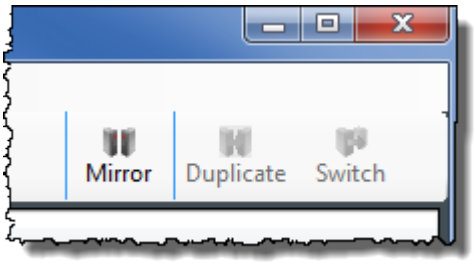
In a redundant processor configuration, you can either:

- use an identical configuration for both the primary and the secondary CP-30s, which is referred to as “mirrored mode”, or
- use independent configurations for the primary and secondary CP-30s.

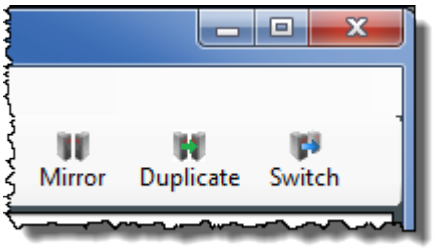
By default, a single mirrored configuration will be created. This can then be downloaded to the primary and secondary processors in turn.

A mirrored configuration is simpler to manage, however sometimes independent primary and secondary configurations will be required. For example, if the two CP-30s need to be accessible via Ethernet (which is a common requirement), then they will need independent IP addresses and therefore separate (non-mirrored) configurations.

To create independent primary and secondary configurations, it is necessary to switch off mirrored mode, by clicking the *Mirror* toolbar button.




**Mirror mode on** – same configuration for primary and secondary processors.




**Mirror mode off** – independent configurations for primary and secondary processors.

With mirror mode switched off, Toolbox PLUS+ will maintain two separate configurations. At any one time, you will be editing either the Primary or the Secondary configuration. The configuration being edited is indicated by:

- one of the CP-30 modules being highlighted in the module list, and
- “Primary” or “Secondary” indicator in the status bar, and
- “Primary” or “Secondary” in the title bar of the *RTU Properties* dialog.

It is important to keep track of which configuration you are editing!

In non-mirrored mode, there are now two additional toolbar buttons:

- **Duplicate:** This copies the primary processor configuration, with the exception of the Port 1 Ethernet settings, to the secondary processor.
- **Switch:** This button switches between editing the primary and secondary configurations. There is also a *Switch* button on many of the *RTU Properties* pages which performs the same function.

If mirrored mode is subsequently re-enabled, the secondary configuration will effectively be erased; the primary configuration will be what is used on both processors.

### 10.1.4 Detecting Status in Logic

When the standby processor takes over control, the transition is largely seamless. Logic execution and I/O processing will continue; normally the only indication that a changeover has occurred is a short delay while the failure is detected (500ms) and the standby initialises I/O scanning.

However, often it is useful to record when changeovers occur, and possibly perform different actions depending on whether the primary or secondary processor is active.

The KF_GET_PROCESSOR function block is useful here. This returns the slot number of the currently active processor. By comparing this to the configured slot numbers of the primary and secondary processor you can determine which is running, and then trigger any actions that may be required.

Another useful function block is KF_GET_MODULE_OK, which can be used to verify that the standby processor is still functional, and raise an alarm if it is not.

### 10.1.5 Shared IP Address

It is often desirable to be able to address each CP-30 individually for maintenance purposes, yet have a SCADA system address the RTU as a whole without necessarily knowing which CP-30 is active.

This can be achieved by using the CP-30's shared IP address feature.

The primary and secondary CP-30s are configured with individual IP address (say, 192.168.0.1 and 192.168.0.2). A third IP address (e.g. 192.168.0.10) is then configured via the *RTU Properties* dialog – see RTU Properties – Redundancy.

The currently active processor will then respond to any requests received on its actual IP address or the shared IP address. The standby processor will ignore the shared IP address, at least until such time as it takes over control.

The SCADA system should be configured to poll the shared IP address. A maintenance tool such as Toolbox PLUS+ can access the individual CP-30s (e.g. to load a new configuration) using their individual IP addresses.

### 10.1.6 Downloading Configurations

For a mirrored configuration, the procedure for downloading the configuration to the RTU is as follows:

1. Connect Ethernet or serial cable to the primary CP-30.
2. Connect to the RTU in Toolbox PLUS+ and download configuration.
3. Connect Ethernet or serial cable to the secondary CP-30.
4. Connect to the RTU in Toolbox PLUS+ and download configuration.

For a non-mirrored configuration, the procedure is as follows (assuming an Ethernet connection):

1. Select the Primary configuration in Toolbox PLUS+

2. Connect to the RTU in Toolbox PLUS+ (which will use the primary CP-30's IP address) and download configuration
3. Select Secondary configuration in Toolbox PLUS+
4. Connect to the RTU in Toolbox PLUS+ (which will use the secondary CP-30's IP address) and download configuration

For more information on downloading configurations see Downloading Configurations.

## 10.2 Redundant Power Supplies

Two PS-xx power supplies can be plugged into a backplane and both will run normally, sharing the power load. If one power supply is removed or fails, the other power supply will supply the complete power load. An ISaGRAF program is not required to manage the power supplies.

When two power supplies are present on the backplane, one power supply can be hot swapped while the RTU is still running. This does not cause any interruption to the processor or inputs and outputs.

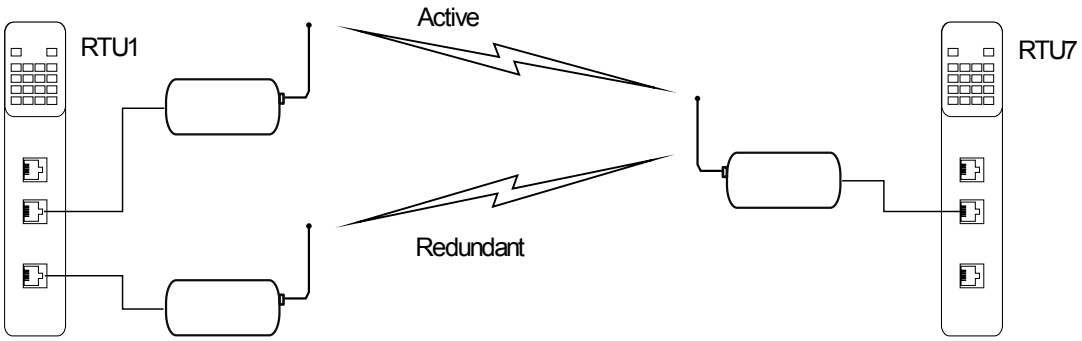
To determine the Total Current supplied by both power supplies to the RTU modules and batteries, the current load for each power supply (SLssPS11AI3.value) can be read and the two figures are totalled.

## 10.3 Redundant Communications

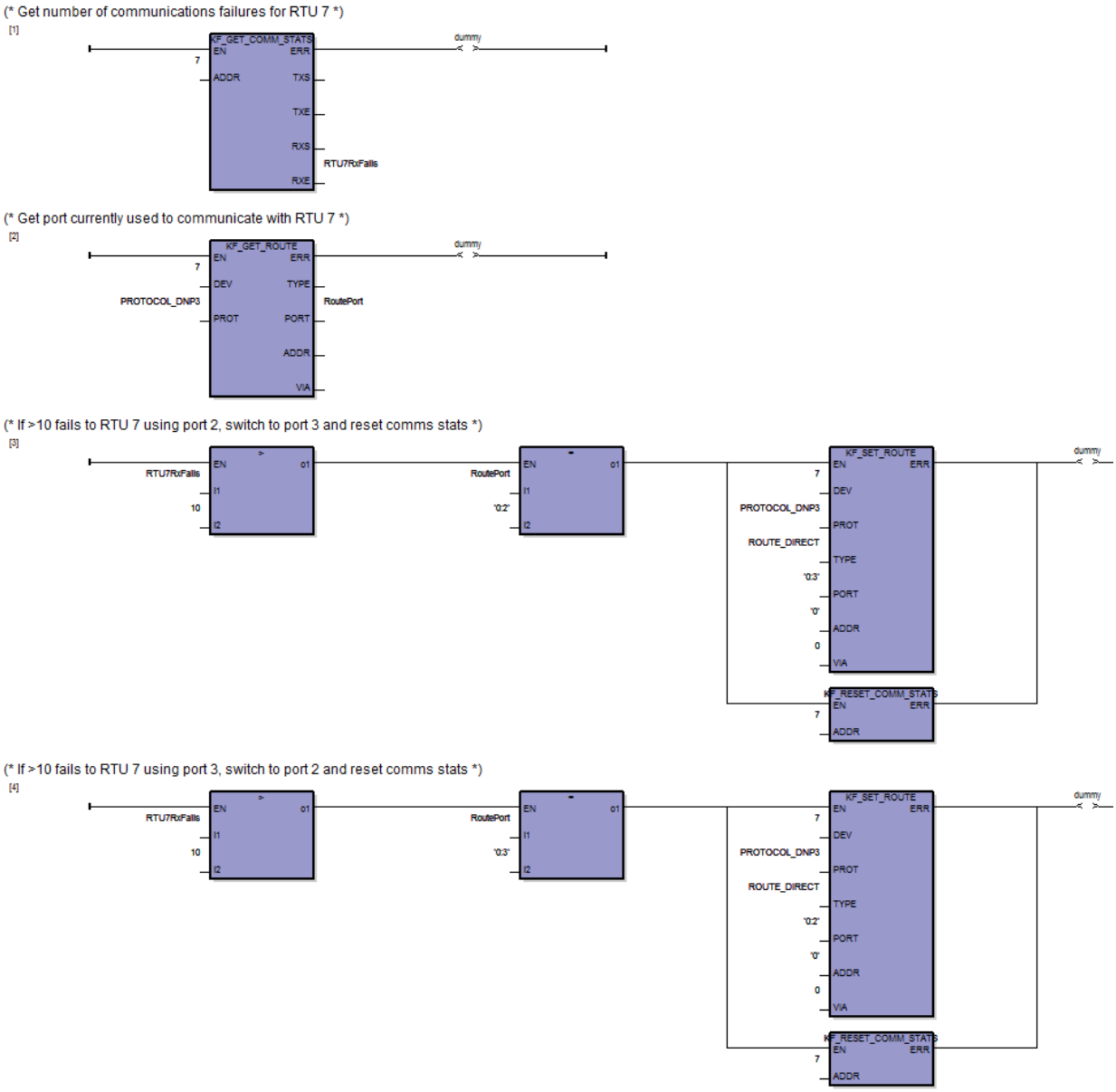
It is possible to change the port and/or route used to communicate with a remote RTU if there is a communications failure.

Note: communications ports on the standby processor cannot be used for redundant communications since these ports will only operate when the standby processor changes into the active processor.

The example below shows how to use CP-30 port 2 as the active port and CP-30 port 3 as the redundant port. Initially RTU1 is configured with a direct route to RTU7 using port 2. The ladder diagram below shows how to swap ports after 11 failed communication attempts to RTU7. The configuration will keep switching between ports if communications continue to fail.



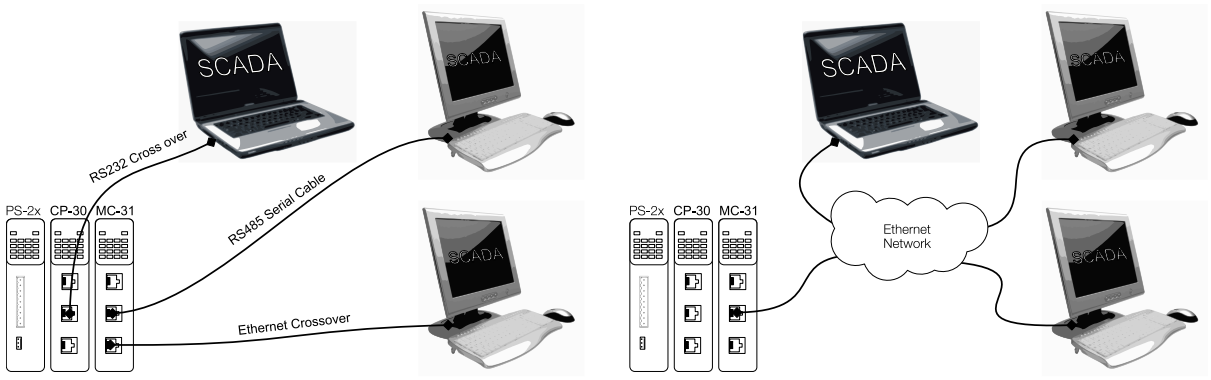

A number of custom function blocks are used below. Please see the topic ISaGRAF Function Blocks for function block details and parameter types. In this example the DNP3 protocol is used; however the technique is equally applicable other protocols.




In this example, traffic was switched from one port to another if a failure was detected. The same method could also be used to switch from a direct to an indirect route. For example, if a direct radio link between two distant RTUs was not available then the sending RTU could fall back to forwarding traffic via an intermediate RTU. This application would use very similar logic, but would change the *TYPE* and *VIA* parameters to KF_SET_ROUTE, rather than *PORT*.

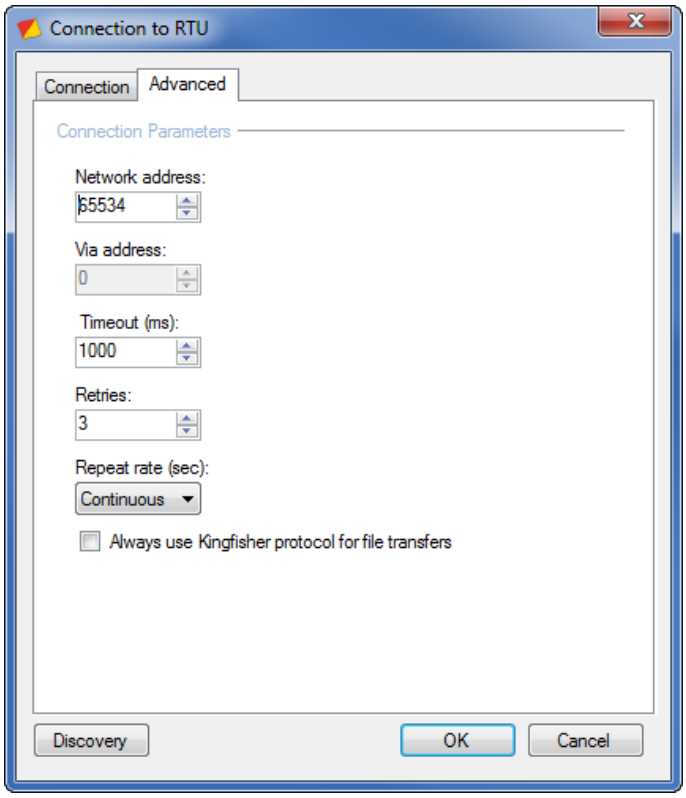
## 10.4 Redundant PCs

Two or more PCs running SCADA or Toolbox PLUS+ software can be connected to the one RTU as illustrated below. All the PCs can poll the same data and set the same outputs. If one PC fails, the other PCs will continue to operate normally.




Each PC can be assigned its own RTU port or all the PCs can share one Ethernet port by using an Ethernet Network. Note: a CP-30/MC-31 Ethernet port can handle communications with up to four devices or RTUs simultaneously.

All the PCs can run the same SCADA software configurations with one exception. Each PC must be assigned a unique RTU address in the range 65521-65535 to prevent communication conflicts. To configure this, click the *Connection* toolbar button. The example below configures Toolbox PLUS+ to use address 65534. Toolbox PLUS+ uses address 65535 by default.



# 11. Download

## 11.1 Overview

Toolbox PLUS+ can be used to create a configuration for an RTU while "offline", i.e. without actually having the RTU connected. Ultimately, however, it is necessary to connect to the RTU in order to download the newly created configuration and logic programs to the RTU.

The general workflow when configuring an RTU is as follows:

1. Define configuration in Toolbox PLUS+ (define module layout, adjust settings)
2. Define logic programs in ISaGRAF Workbench
3. "Build" the configuration and logic. The result is a set of files which the RTU can understand. This will be done automatically if the configuration or logic have changed. It can also be triggered manually, using the *Build* toolbar button. See also ISaGRAF Overview.
4. Download these compiled files to the RTU.
5. Restart the RTU (this will happen automatically)

Toolbox PLUS+ can also be used to download new firmware to the RTU.

## 11.2 Firmware Compatibility

Various versions of Toolbox PLUS+ and firmware have been released. To ensure correct operation, the Toolbox PLUS+ version must be compatible with the firmware version in the RTU. The following table summarises the relationship between Toolbox PLUS+ version and firmware version.

| Toolbox PLUS+ Version | Date | Minimum Firmware Version |
|---|---|---|
| 3.8.4 | Dec 2010 | 2540 |
| 3.8.9 | Oct 2011 | 2676 |
| 3.9.2 | Apr 2012 | 2676 |
| 3.10.1 | Nov 2012 | 2874 |
| 3.11.0 | Aug 2013 | 2874 |

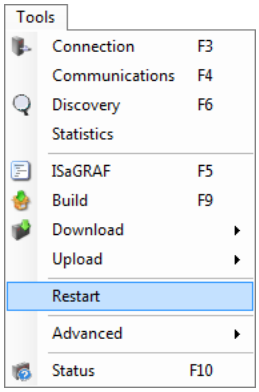
Note that Toolbox PLUS+ can be used to maintain RTUs with firmware older than that specified above, including upgrading firmware and updating existing projects. However, new projects cannot be created because they will contain features (e.g. new function blocks) that are not understood by the older firmware. In order for Toolbox PLUS+ to be fully functional, the RTU firmware should be upgraded to at least the level indicated in the table.

Note: The current version of Toolbox PLUS+ does not support downloading configurations to RTUs with firmware version 2150 or older. It can still be used to upgrade the RTU firmware, however.

## 11.3 RTU Restart

Restarting the RTU resets all the variables to their initial values by default (although individual variables can be configured to retain their value during a restart) and resets all other settings to their power up defaults. A restart retains event logs, firmware, logic and the RTU configuration.

The RTU will be automatically restarted following download of firmware or a new configuration. If required it can also be manually restarted from Toolbox PLUS+.



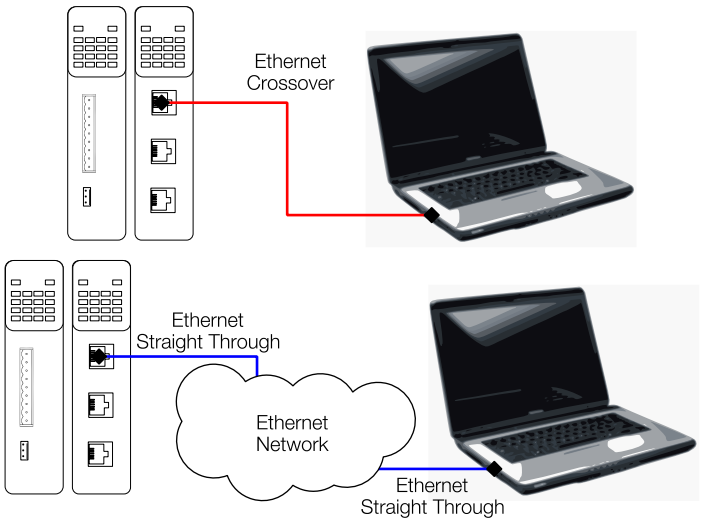

Select the RTU name in the navigation pane

Select **Tools → Restart**

## 11.4 Connecting to an RTU

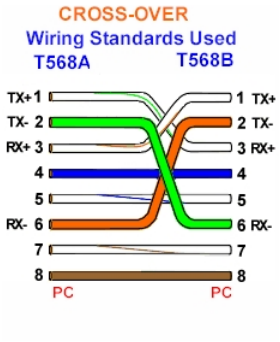
### 11.4.1 Cables

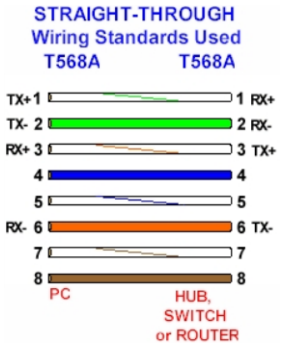
To connect a PC directly to an RTU, an Ethernet *crossover* cable is normally required as illustrated below, although some modern PCs may also work with a straight through cable. If connecting to an Ethernet hub or switch, a straight through cable or a crossover cable can be used.

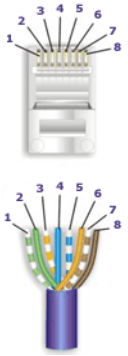



By default, the IP address of the RTU's main Ethernet port is set to 192.168.0.1. This may not be suitable for connecting via an existing Ethernet network – check with your network administrator. If this is the case then you will need to perform the initial setup using a direct crossover cable connection. The CP30 should not be connected to the existing Ethernet network until it has been configured with an IP address that is compatible with the network.

If required, the CP30 can be reset it to its factory settings (including setting its IP address to 192.168.0.1) by removing the link on the rear edge of the module next to the backplane connector, waiting 5 minutes, then replacing it.

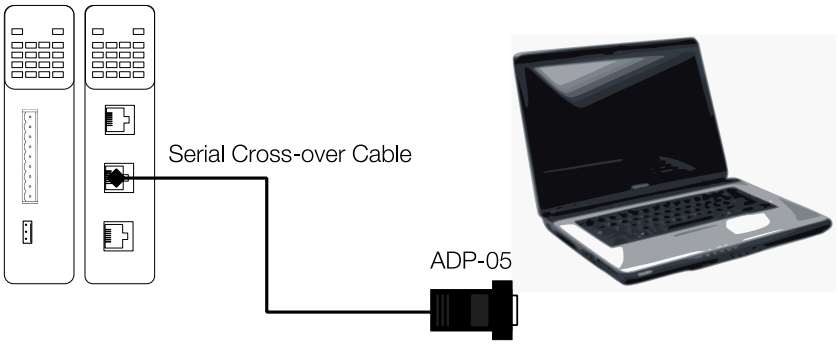


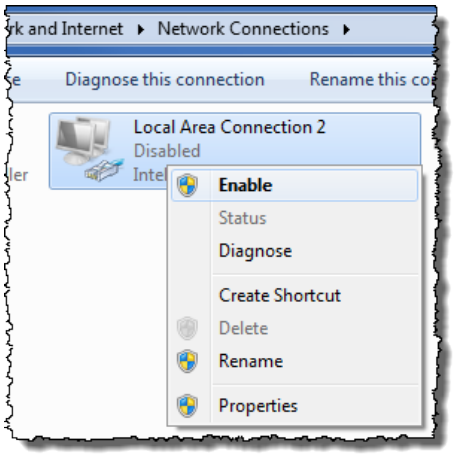

The above cables are industry standard and are readily available. Once a cable is installed, the PC's LAN port needs to be setup as detailed in the LAN Port Setup.

If the RTU has a Serial option port and the port has already been configured (by first using Ethernet communications), then Toolbox PLUS+ can use Serial communications to the RTU as illustrated below.




### 11.4.2 LAN Port Setup

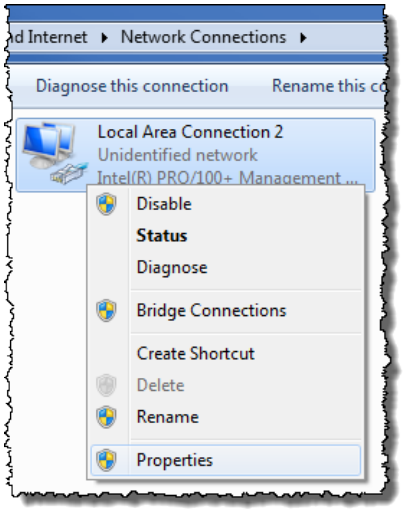
To initially communicate with the RTU, the PC's LAN port must be enabled and setup to communicate with the RTU's default IP address (192.168.0.1) as detailed below.



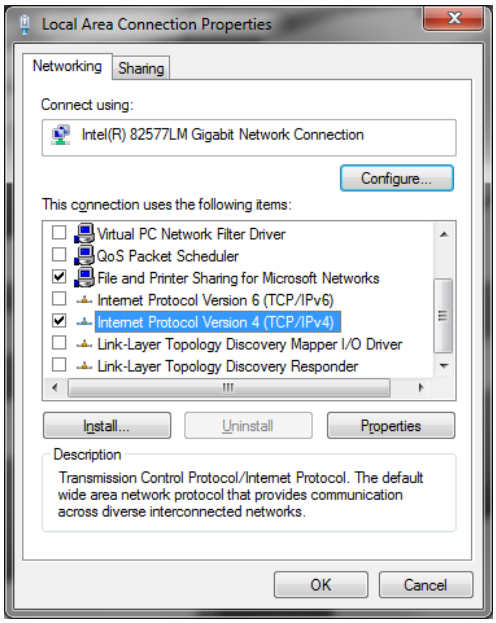

For Windows 7: Open Control Panel, select *Network and Sharing Center*, then *Change adapter settings.*

For Windows XP: Open Control Panel, select *Network Connections*.

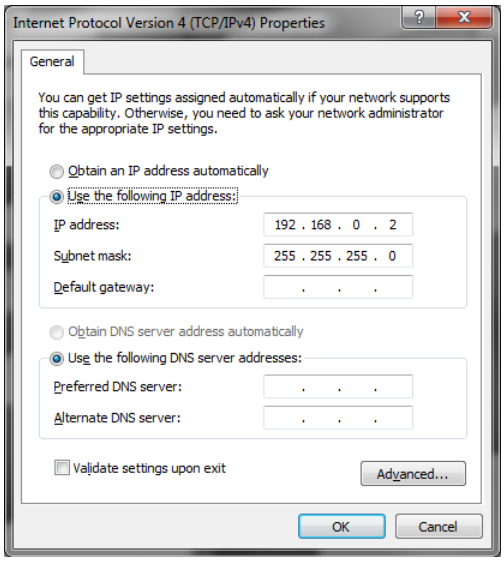
This will show a list of the available network adapters in your PC. Identify the one corresponding to the RTU connection and confirm that it is enabled. If it is marked "Disabled", right click and select *Enable*.




Once the LAN port is enabled, right-click the LAN port again and select *Properties*.

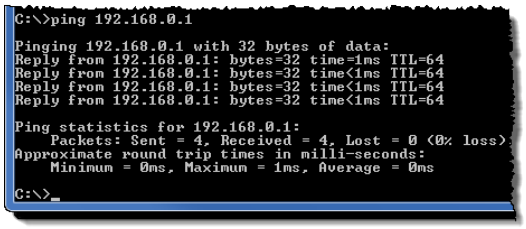



Select *Internet Protocol (TCP/IPv4)* and then click *Properties*.


Internet Protocol Version 4 (TCP/IPv4) Properties
General
You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.
Obtain an IP address automatically
Use the following IP address:
IP address: 192 . 168 . 0 . 2
Subnet mask: 255 . 255 . 255 . 0
Default gateway:
Obtain DNS server address automatically
Use the following DNS server addresses:
Preferred DNS server:
Alternate DNS server:
Validate settings upon exit
Advanced...
OK
Cancel

Set the IP address and subnet mask as shown. Note that any unused address in the range 192.168.0.2 to 192.168.0.254 can be used for the PC.

Press OK to save the settings.




Verify the connection by starting a Windows command window and typing: *ping 192.168.0.1*

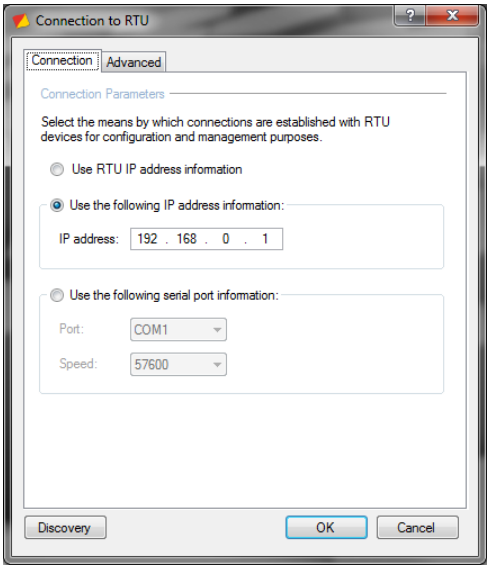
The results should indicate that replies are being received from the RTU. If not, check your cables and LAN settings.

### 11.4.3 Connection Parameters

The *Connection* toolbar button is used to define how Toolbox PLUS+ will communicate with the RTU.

Normally, Toolbox PLUS+ will use the RTU address information (e.g. IP address) as specified in the configuration. However, sometimes you will need to override this – e.g. if you have changed the IP address in the configuration then you will need to tell Toolbox PLUS+ to connect using the old address in order to load the new configuration.

The *Connection to RTU* dialog has the following settings.



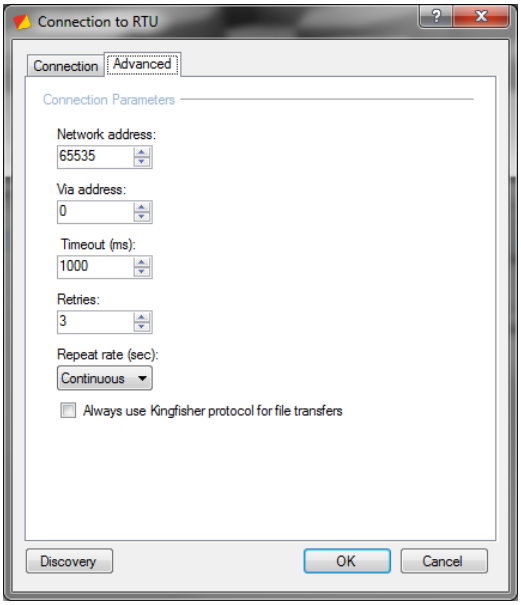

**Use RTU IP address information:** When selected, Toolbox PLUS+ uses the IP address configured for the processor Port 1.

**Use the following IP address information:** When selected, Toolbox PLUS+ uses the specified IP address.

**Use the following serial port information:** When selected, Toolbox PLUS+ uses the specified serial port (COM1, COM2 …) and speed (9600, 19200, 38400, 57600 or 115200 bps) to communicate with the RTU using the Kingfisher protocol.

Note: If the CP30's IP address is unknown then it can be reset to the factory default of 192.168.0.1 by removing the link on the rear of the module, waiting 5 minutes, then replacing the link.

The *Advanced* tab contains additional settings:




**Network Address:** (1-65535) Because the PC is treated like another RTU in the network, it must have its own unique address. Addresses 65521 to 65535 are reserved for Toolbox PLUS+. If two or more PCs running Toolbox PLUS+ are connected to the same RTU, each PC should have its own unique address to avoid communication conflicts.

**Via address:** (0-65535) When using a network of RTUs, this is the address of the first RTU to communicate through. Address 0 [default] disables this function.

**Timeout (ms):** The time to wait for the RTU to respond to messages. This setting may need to be lengthened when communicating using a slow serial link (e.g. a setting of at least 4000ms when using 1200 baud)

**Retries:** The maximum number of attempts (after the first attempt) Toolbox PLUS+ will make at sending a message to the RTU if the previous attempts have failed.

**Repeat Rate:** This allows a delay to be inserted between message attempts.

**Always use Kingfisher protocol for file transfers:** When enabled, Toolbox PLUS+ uses the Kingfisher protocol when downloading firmware and configurations over Ethernet or serial connections. When disabled, Toolbox PLUS+ uses the SSH internet protocol, which is significantly faster.

Using the SSH protocol is normally preferable. However, Kingfisher protocol will need to be used if:

- A serial connection is used to connect to the RTU
- A configuration or firmware is being downloaded to a CP-30 via an MC-31 Ethernet port
- A configuration or firmware is being downloaded to a CP-30 via another RTU

### 11.4.4 Discovery

The *Discovery* toolbar button tells Toolbox to try to locate the RTU on the local Ethernet or serial network.

For Ethernet, Toolbox PLUS+ will send out a broadcast packet on the local subnet and collect any replies from RTUs. The RTU address and IP address will be displayed for all RTUs that were found.

Note that this will only work if the RTU's current IP address is set to a value which is compatible with the local subnet. For example, if the RTU and your PC are connected to the 192.168.0.x LAN, but the RTU's IP address is set to 10.2.3.12, then it will not be detected.

### 11.4.5 Status Check

Once the RTU's IP address and RTU address have been determined, the *Status* toolbar button can be used to check that communications are working. See Status for more details.

## 11.5 Downloading Configurations

### 11.5.1 Connection

In order to download a new configuration, the required project must be open in Toolbox PLUS+, and the required RTU selected.

*Main CP30 Ethernet port*

The simplest and fastest method of downloading a configuration to the CP30 is to use the main Ethernet port on the CP30.

Before attempting to download the configuration, ensure that the connection parameters are set correctly. In particular:

- If the main port IP address set in the configuration is the same as the CP30's current IP address then select *Use RTU address information*. If the new configuration will change the main port IP address then select *Use the following IP address information*, and enter the CP30's current IP address.
- On the *Advanced* tab, ensure that the *Always use Kingfisher protocol* checkbox is not ticked. The SSH protocol will then be used.

*Other CP30 Ethernet ports*

It is also possible to update a CP30's configuration via a connection to a T3 Ethernet option card (installed as port 2 or 3 on a CP30). The CP30 must have been previously configured to enable the additional port and set its IP address.

Before attempting to download the configuration, ensure that the connection parameters are set correctly. In particular:

- Select *Use the following IP address information*, and enter the current IP address of the T3 Ethernet port.
- On the *Advanced* tab, ensure that the *Always use Kingfisher protocol* checkbox is not ticked. The SSH protocol will then be used.

*Other connection options*

With some caveats, it is also possible to update a CP30's configuration via an MC31 port, or a serial connection, or via another RTU. The CP30 must have been previously configured to enable the port and set its IP address or serial parameters.

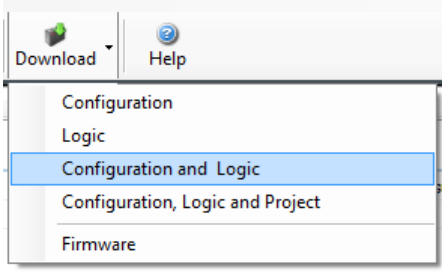
Note: It is not possible to change the RTU's address using these connection methods. The address (1-65520) set in the RTU Properties dialog must match the RTU's current address.

Before attempting to download the configuration, ensure that the connection parameters are set correctly. In particular:

- If connecting to an MC31 Ethernet port, select *Use the following IP address information*, and enter the current IP address of the MC31 port.
- If connecting to a serial port, select *Use the following serial port information* and enter the current baud rate set for the serial port. (Note that the serial configuration downloads are only supported if the RTU port is configured for 8 data bits, no parity, 1 stop bit.)

- If connecting via an intermediate RTU, be sure to use the IP address or serial baud rate of the intermediate RTU (not the destination RTU) for the above settings. On the *Advanced* tab, set the *Via address* field to the address of the intermediate RTU.
- On the *Advanced* tab, ensure that the *Always use Kingfisher protocol* checkbox is ticked. The Kingfisher (KF) protocol will then be used.

### 11.5.2 Download




To download a new configuration to the RTU, click the *Download* toolbar button, then *Configuration and Logic*.

This command will download the RTU configuration (module definition and settings), and any logic programs (including Dictionary variables).

(It is also possible to download just the configuration, or just the logic, but this is generally not useful unless you are connected via a very slow communications link.)

The following operations will then occur automatically:

1. If necessary, the configuration and/or logic will be rebuilt.
2. The configuration files will be downloaded to the RTU
3. The RTU will extract and process the files
4. The RTU will restart
5. Toolbox PLUS+ will reconnect to the RTU and confirm that the update was successful.

It is also possible to download the entire project file to the RTU, by selecting *Configuration, Logic and Project*. The project can then be retrieved from the RTU (using the *Upload Configuration* option on the *Tools* menu) at a later date, where it can be viewed or modified.

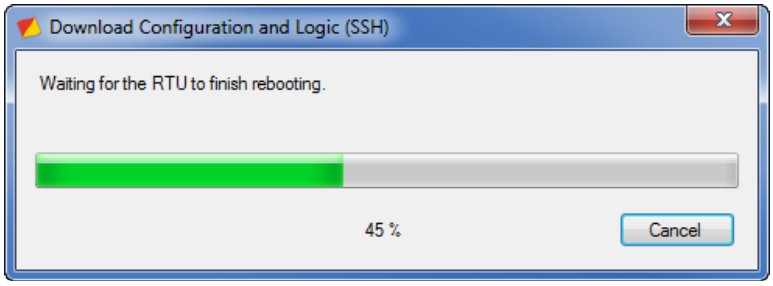
If the project file is not saved to the RTU then the *Upload Configuration* function can still reconstruct the basic configuration in Toolbox PLUS+, but the logic programs will not be able to be loaded.

The downside of storing the project file on the RTU is that it can be quite large, so it will use up space on the RTU and increase the time taken to download new configurations.

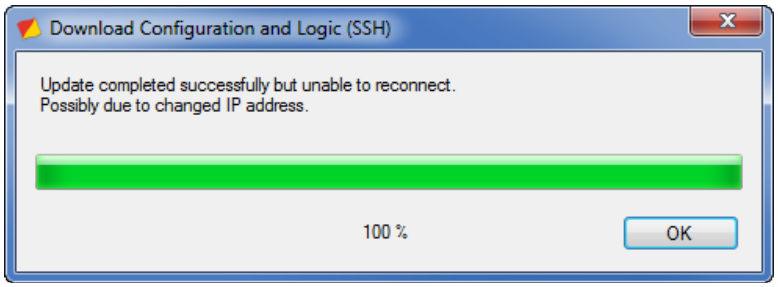
### 11.5.3 Reconnection

If the newly downloaded configuration has changed the communications settings on the RTU (e.g. IP address, serial port baud rate) then it is possible that after the RTU restarts, Toolbox PLUS+ will be unable to reconnect.

If this is the case then the download progress dialog will stay on screen for about 3 minutes:




At this point, the configuration update has already completed, so you can safely press *Cancel*. Alternatively, if you wait then a timeout message will eventually be displayed:


Download Configuration and Logic (SSH)
Update completed successfully but unable to reconnect.
Possibly due to changed IP address.
100 %
OK

To verify the new configuration, click the *Connection* toolbar button then select the *Use RTU address information* option. Then press the *Status* toolbar button to check that you can communicate with the RTU using the new settings.

## 11.6 Downloading Firmware

### 11.6.1 Overview

Firmware can be downloaded into a processor or communications module to upgrade functionality. The firmware version that is currently in the CP-30, G30, or MC-31 can be checked using the Status command.

Note that the CP-30 and MC-31 use identical firmware. The G30 firmware is slightly different, however.

When upgrading firmware, it is recommended that all CP-30 and MC-31 modules in an RTU be upgraded to the same version.

Upgrading firmware will preserve the current RTU configuration (including configured IP addresses). However, all logic will be cleared, as will all event logs.

Note: The process of upgrading firmware can take several minutes. To minimise the chance of the upgrade failing, firmware upgrades should be performed using a reliable communications link (e.g. Ethernet) and with a stable power supply to the RTU.

### 11.6.2 Connection

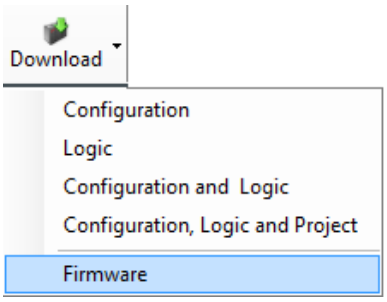
The simplest and fastest method of downloading firmware to a CP30 or MC31 is to use an Ethernet port – normally the main Ethernet port, but you can also use port 2 or 3 if you have a T3 Ethernet option card installed. The RTU must have been previously configured to enable the port and set its IP address (unless the main port is being used with its factory default IP address).

It is also possible to update a CP30's firmware via an MC31 port, or a serial connection, or via another RTU. Note that this method will be significantly slower, and it is not possible to update MC31 firmware using this method.

For MC31, a direct Ethernet connection is required, and the SSH protocol must be used (ensure that on the *Always use Kingfisher protocol* checkbox is not ticked).

Before attempting to download the firmware, ensure that the connection parameters are set correctly, as described in Downloading Configurations.

### 11.6.3 Download




The process for upgrading firmware on a CP-30, G30 or MC-31 is as follows:

If no project is loaded then the *Connection To RTU* window will appear. Specify the IP address of the Ethernet port that firmware will be downloaded to. If a project is loaded then ensure that the correct RTU is selected.

The *Select Firmware File* window will appear. Select the firmware file to download into the CP-30, G30 or MC-31. This will have a name such as **firmware2981.tgz**. For G30, be sure to select the G30 firmware file, i.e. **firmware2981_g30.tgz**

The following operations will then occur automatically:

1. The first part of the firmware package will be downloaded to the RTU
2. The RTU will extract and process the files, then restart
3. Once Toolbox PLUS+ detects that the RTU has restarted, the second part of the package will be downloaded.
4. The RTU will extract and process the files, then restart
5. Once Toolbox PLUS+ detects that the RTU has restarted, the third and final part of the package will be downloaded.
6. The RTU will extract and process the files, then restart.
7. Toolbox PLUS+ will reconnect to the RTU and confirm that the update was successful.

Assuming an Ethernet link, this process will typically take 5-10 minutes to complete.

### *Upgrading Redundant Processor Systems*

For a system with redundant CP-30 processors, you will need to upgrade the firmware for each processor separately.

Note that:

- When upgrading firmware using the default SSH protocol, the upgrade will occur regardless of whether the processor you are upgrading is currently the active or standby processor. In fact, if the CP-30 is the active processor then when it restarts after processing the first part of the upgrade it will come up as the standby, because the other CP-30 will have taken over during the restart. This should not affect the upgrade process.
- If your only connection to the RTU is via an MC-31, then the CP-30 firmware can still be upgraded by selecting the Kingfisher protocol (if you use SSH then you will actually upgrade the MC-31 firmware). Note however that in this case it is essential that the other CP-30 is removed from the RTU during the upgrade. Otherwise, after the first part of the upgrade the other CP-30 will assume control, and the second part of the upgrade package will be directed to it (as the active RTU processor) rather than the CP-30 that is being upgraded. This will cause the upgrade to fail.

# 12. Viewing Data

## 12.1 Status

The *Status* command reads the following information from the selected RTU:

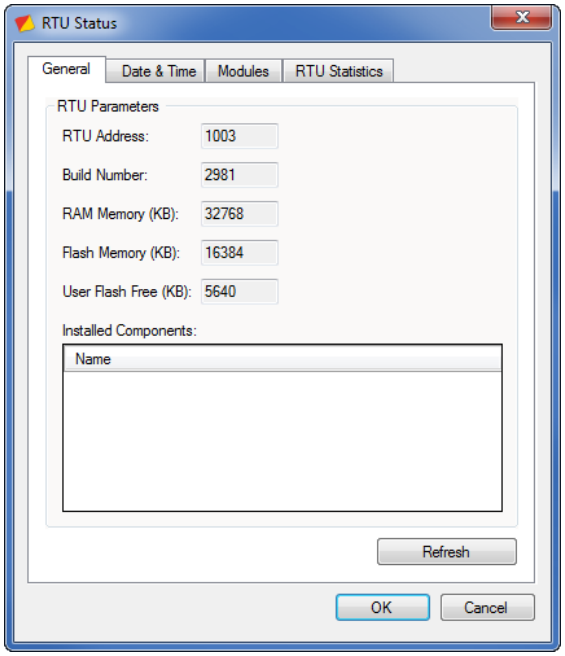
- Basic details including RTU address and firmware version
- Current RTU Date and Time. This can also be set.
- Status of all detected I/O modules
- Communications statistics

Note that in order for the status request to be successful:

- The connection parameters (IP address or serial baud rate) must match the current actual settings in the RTU, and
- The RTU address in the configuration must match the RTU's current actual address.

If RTU's IP address or RTU address are not known then the Discovery command may be used. Failing that, the CP30 can be reset to factory settings (address 1, IP 192.168.0.1) by removing the link at the rear of the module, waiting 5 minutes then replacing it.

To perform a status request, click the *Status* toolbar button. If no RTU is selected then you will be prompted for the IP address to use.




On the *General* tab:

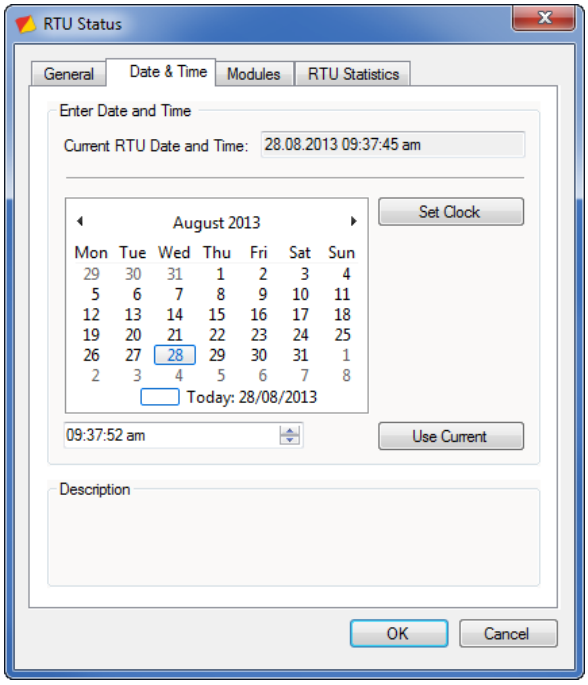
**RTU Address:** The address of the currently connected RTU (1-65520)

**Build Number:** Firmware version number

**User Flash Free (KB):** The amount of free non-volatile memory available in the RTU

**Installed components:** Not used.

**Refresh:** Re-reads the information from the RTU.



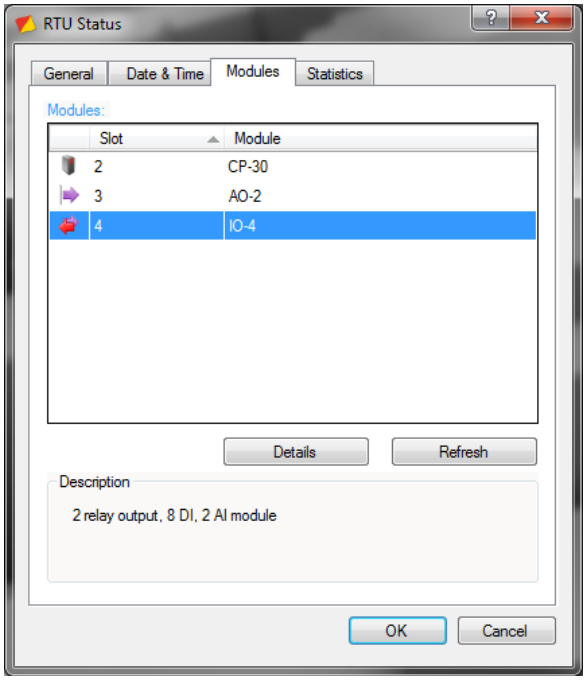

On the *Date & Time* tab:

**Current RTU Date and Time:** The current RTU time is displayed here, updated once per second.

**Set Clock:** Sets the RTU time to that set on the date/time controls.

**Use Current:** Set the date/time controls to the current PC time.

Thus to sync the RTU to your PC's time, you would click *Use Current*, immediately followed by *Set Clock*.

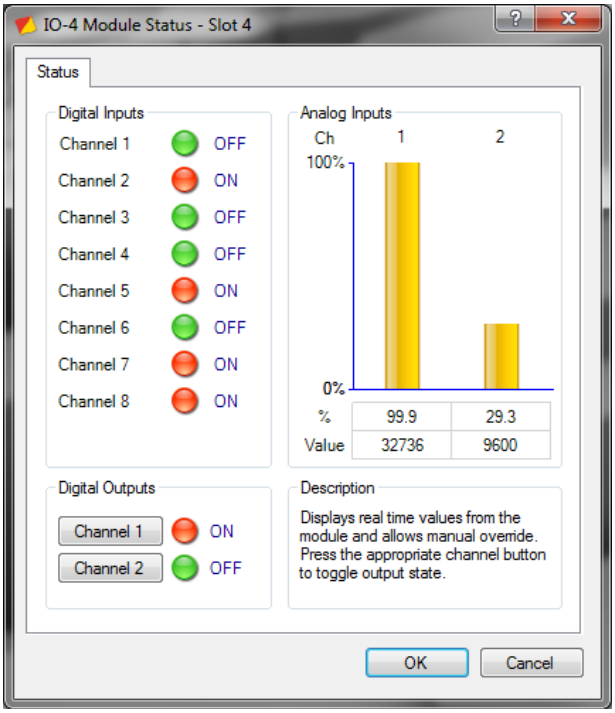



On the *Modules* tab:

**Slot/Module:** Lists all detected I/O modules in the RTU.

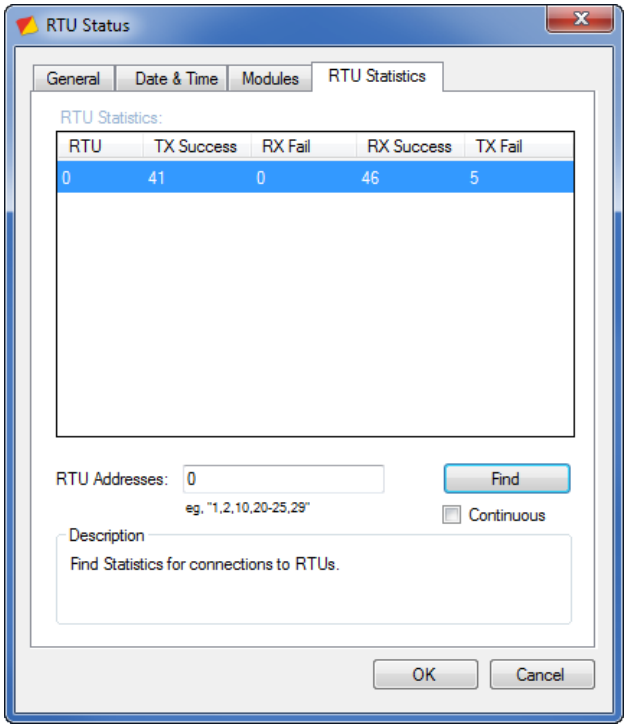
**Details:** Displays context specific details of the currently selected module. The information presented will depend on the type of module.

**Refresh:** Re-scans the RTU for additional modules




Sample module details display:

For an IO-4 module, the module status display shows the state of the 8 digital inputs, current readings for the two analog inputs, and provides buttons for toggling the two digital outputs.




On the *RTU Statistics* tab:

This shows communications statistics recorded by the selected RTU. The statistics are the same as those reported by the KF_GET_COMM_STATS function block.

**RTU Addresses:** Statistics will be shown for communications attempted between the selected RTU and each RTU address in this list. An address of 0 will return aggregate statistics across all RTUs.

**Find:** Update displayed statistics.

**Continuous:** Continuously update displayed statistics.

## 12.2 Event Logs

Event logs allow the RTU to record time and date stamped data. An event log is recorded when:

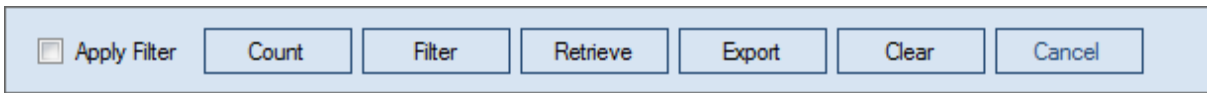
- The KF_EVENT_LOG function block is executed.
- A class 1, 2 or 3 DNP3 variable changes in value. For analog inputs, the change must exceed the configured deadband.
- DNP3 events are received from another RTU, e.g. in response to a DNP3 class 1/2/3 poll request.
- Kingfisher events are received from another RTU, e.g. in response to a Kingfisher protocol event log request
- Events are read from a digital input module that supports Sequence-Of-Events recording (e.g. DI-10)

The number of event logs that can be recorded in the RTU memory is configurable on the RTU Properties dialog.

### 12.2.1 Viewing Event Logs

To retrieve and view event logs:

- Select the RTU name in the navigation pane
- Select *Event Log* in the Stacked Menu Bar
- Select the *Retrieve* button
- Specify whether to retrieve All Events or the number of newest events to retrieve




### 12.2.2 Event Log Buttons

**Apply Filter:** Applies the configured filter settings when retrieving event logs, so that only those matching the filter will be retrieved.

**Count:** Queries the number of event logs currently stored in the RTU and displays the result (to the right of the *Cancel* button).

**Filter:** Configures the filter settings

**Retrieve:** Retrieves event logs from the RTU. This can retrieve all events or a configurable number of events.

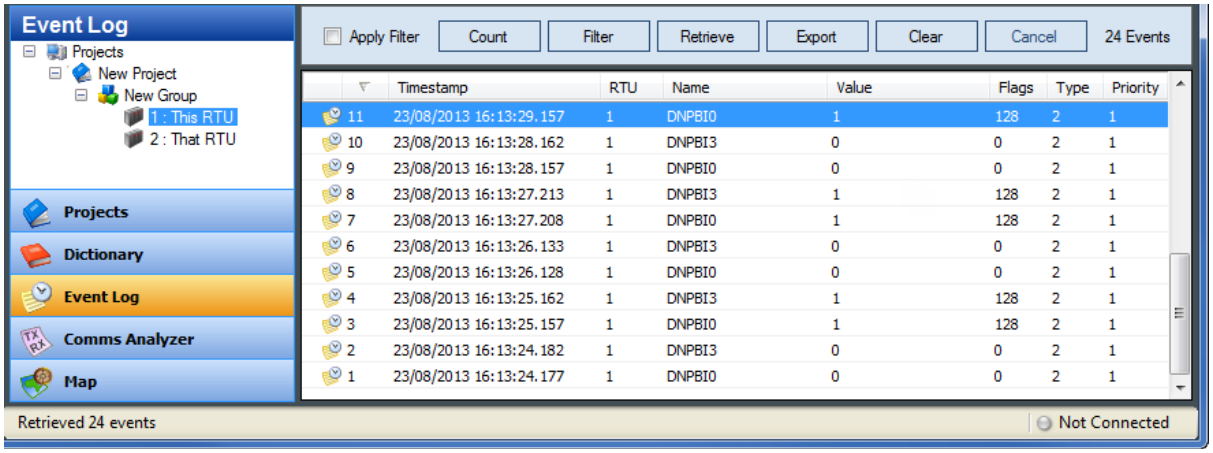
**Export:** Saves the uploaded event logs into a CSV file (can be opened using Microsoft Excel).

**Clear:** Clears all the event logs in the RTU.

**Cancel:** Only available while retrieving event logs. Stops the uploading of event logs.

### 12.2.3 Event Log Parameters

Retrieved event logs are listed in column format, e.g.:




The various parameters that make up an event log are listed below.

**Id:** Unique index assigned to each event log by the RTU.

**Timestamp:** Date and time of the event log.

**RTU:** Address of the RTU that created the event log.

**Name:** Name of the variable that was logged.

**Value:** Value of the variable when logged.

**Flags:** Additional information available when using DNP3.

**Type:** Configured type of the event log. For DNP3, this is the event variation.

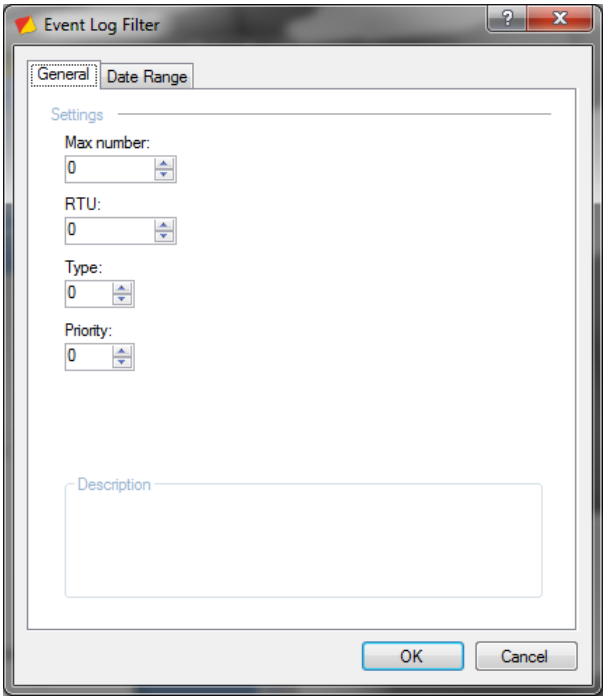
**Priority:** Configured priority of the event log. For DNP3, this is the event class.

### 12.2.4 Event Log Filter

To configure the Event Log Filter

- Select the RTU name in the navigation pane
- Select *Event Log* in the Stacked Menu Bar
- Select the *Filter* button
- Specify the General settings of the event logs to upload
- Select the *Date Range* tab to specify time and date settings of the event logs to upload
- Enable the filter by ticking *Apply Filter*.
- Select the *Retrieve* button to upload event logs that correspond to the filter settings

The event filter settings are shown below:




**Max Number:** (0 – 65535) Maximum number of event logs to retrieve (0=all)

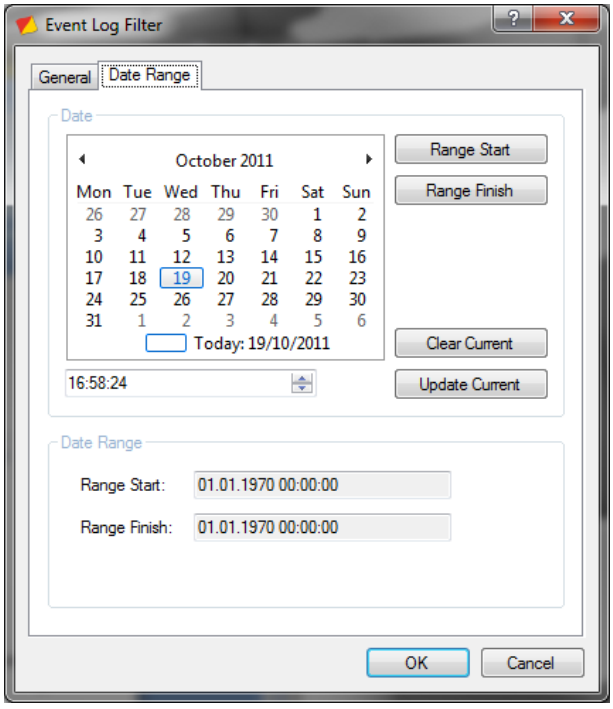
**RTU:** (1-65520) Only event logs created by this RTU will be retrieved (0=all).

**Type:** (0-31) Only event logs matching this Type setting will be retrieved.

**Priority:** (0-7) Only event logs matching this Priority setting will be retrieved.

**Range Start:** The date and time of the oldest event log to retrieve.

**Range Finish:** The date and time of the newest event log to retrieve.




**Range Start/Range Finish:**
Setting these dates allows users to isolate a time period. Users can select a date in the calendar, then select the buttons to update the start/finish date.

**Clear Current:**
Sets the selected range to the start of the event log index

**Update Current:**
Sets the selected rage to the current date

**Range Start/Range Finish**
These textboxes are the summary of the requested period. Values of “01.01.1970” indicate “any date”.

## 12.3 Communications

When troubleshooting RTU systems, it is frequently very helpful to be able to capture the actual message packets that are being sent and received. If a problem with the RTU is suspected the capture files can be forwarded to our support team for analysis.

Two main tools are generally used for this purpose:

- Toolbox PLUS+ *Comms Analyzer* feature
- *Wireshark* – a free and widely used packet capture PC application.

In general terms, Wireshark is more powerful, but Comms Analyzer can be used in a wider range of scenarios.

### 12.3.1 Comms Analyzer

#### *Overview*

The Toolbox PLUS+ Comms Analyzer works as follows:

1. The selected RTU is instructed to start capturing communications messages on a particular port.
2. From that point on, each time a message is sent or received, the information is returned to Toolbox PLUS+, using a separate RTU port to the one being monitored.
3. Toolbox PLUS+ saves the received packets and allows them to be viewed.

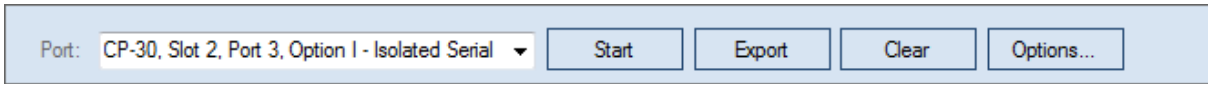
The main limitations of Comms Analyzer are:

- It is not possible to capture traffic on the physical RTU port that is being used to connect to Toolbox PLUS+
- Only basic packet decoding is performed (Wireshark's packet decoding is much more extensive)
- There is a performance impact on the RTU being monitored and on the communications link between that RTU and Toolbox PLUS+, while capture is enabled.

#### *Using Comms Analyzer*

To use Comms Analyzer, first select the RTU in the navigation pane, then click *Comms Analyzer* in the Stacked Menu Bar.

The controls for the Comms Analyzer are at the top of the workspace:



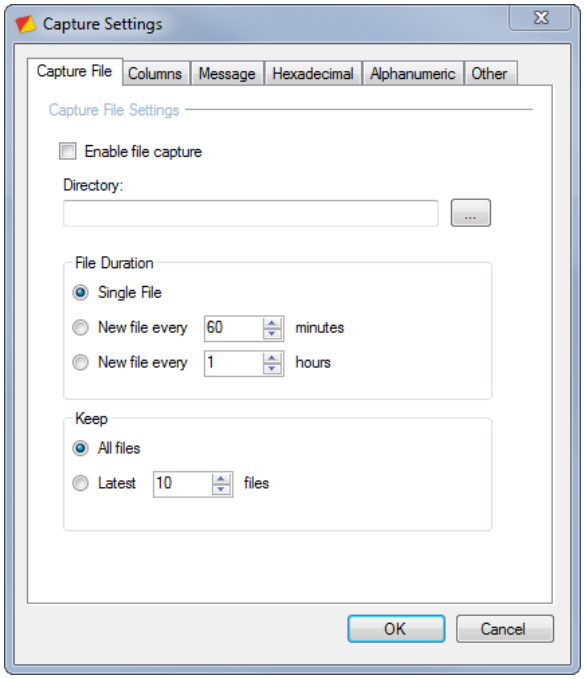

**Port:** This contains a list of all defined ports in the RTU. Select the one for which you want to capture traffic. Remember that it is not possible to capture traffic on the port that Toolbox PLUS+ is using to communicate with the RTU.

**Start:** Start capture on the selected port. This button changes to **Stop** while capture is in progress.

**Export:** Saves some or all the captured packets in CSV format. The generated file can then be opened in a text editor or spreadsheet application.

**Clear:** Deletes all captured packets.

**Options:** Allows a number of settings to be adjusted:




The following tabs are available:

**Capture File:** If *Enable file capture* is ticked then packet data will be continuously saved to capture files. This is useful if you need to leave the Comms Analyzer running for a long period of time in order to capture a rare event.

**Columns:** Allows some of the columns to be hidden

**Message:** Specifies how the various components listed in the *Message* column are separated

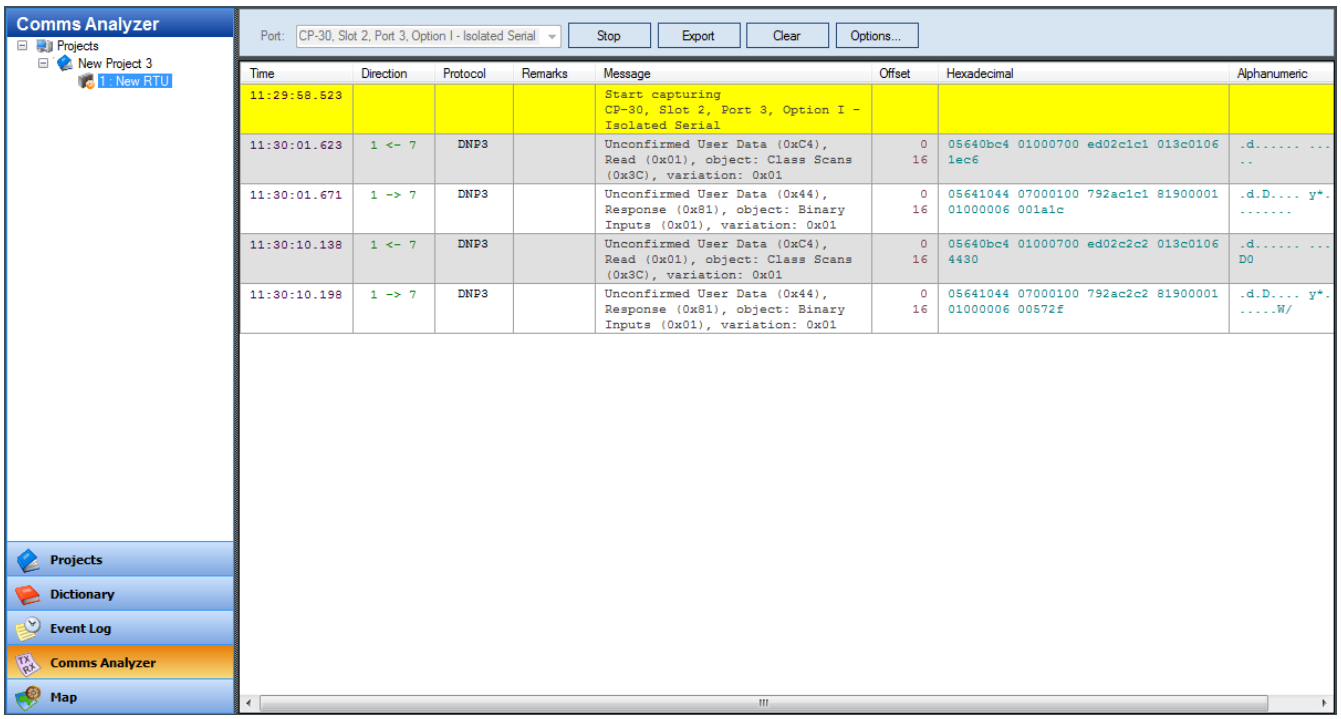
**Hexadecimal:** Specifies how the hexadecimal message data is formatted

**Alphanumeric:** Specifies how the alphanumeric message data is formatted

**Other:** Various other cosmetic display adjustments

To start capturing, click *Start*. Notice that while capture is active an animated overlay is shown on the RTU's icon in the navigation pane.

A typical Comms Analyzer display is shown below:



This shows:

- an entry indicating the time at which capture was started, and the port being monitored
- two incoming DNP3 polls from a master (address 7) to the local RTU (address 1), and the responses sent by the RTU.

The *Message* column provides a basic decode of the message. Note that it does not necessarily list everything you need to know about the message. For example, for a DNP3 message it lists the header fields and the first data object only.

The yellow highlight indicates that the packet is selected. By clicking on rows in conjunction with the Shift and Ctrl keys, you can select a range of packets, which can then be exported using the *Export* button.

### 12.3.2 Wireshark

*Overview*

Wireshark is a PC-based application which can capture all traffic sent or received by the PC's Ethernet interfaces. It has extensive packet filtering capabilities, and can decode many different protocols.

The main limitations of Wireshark are:

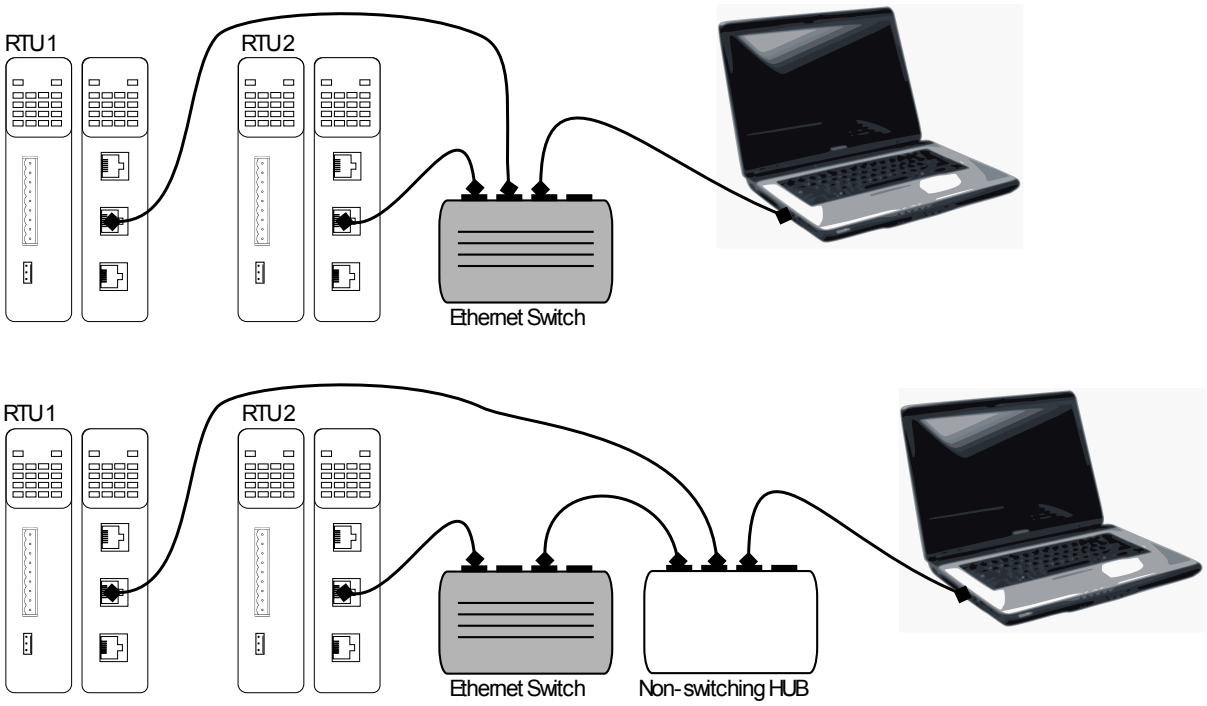
- It can only capture Ethernet traffic.
- It can only capture traffic that is received or sent by the PC. Normally this means that it can only capture traffic sent to or by the PC. However, by using an Ethernet **hub** it is possible for the PC to "snoop" on Ethernet traffic sent from one RTU to another.

*Snooping*

In a modern Ethernet network, each device (PC, RTU, etc.) is normally connected to an Ethernet **switch** or **router**. The switch or router examines each packet that passes through it and directs it to the appropriate destination device. This means that if two RTUs and a PC are all connected to an Ethernet switch or router, then the communications between the two RTUs will be invisible to the PC, which means that Wireshark will not be able to capture them.

However, there is an older technology which can be used to connect devices on an Ethernet network, called a **hub**. Hubs broadcast each received packet to all connected devices. The devices will normally discard anything not addressed to them, with the exception of tools such as Wireshark, which can capture them.

The following diagram shows how a hub can be temporarily added to a network to allow Ethernet traffic sent between two RTUs to be captured.




In the top diagram, RTU1 and RTU2 are connected via an Ethernet switch, so the PC will not receive (and will therefore not be able to capture) any traffic except that addressed to it.

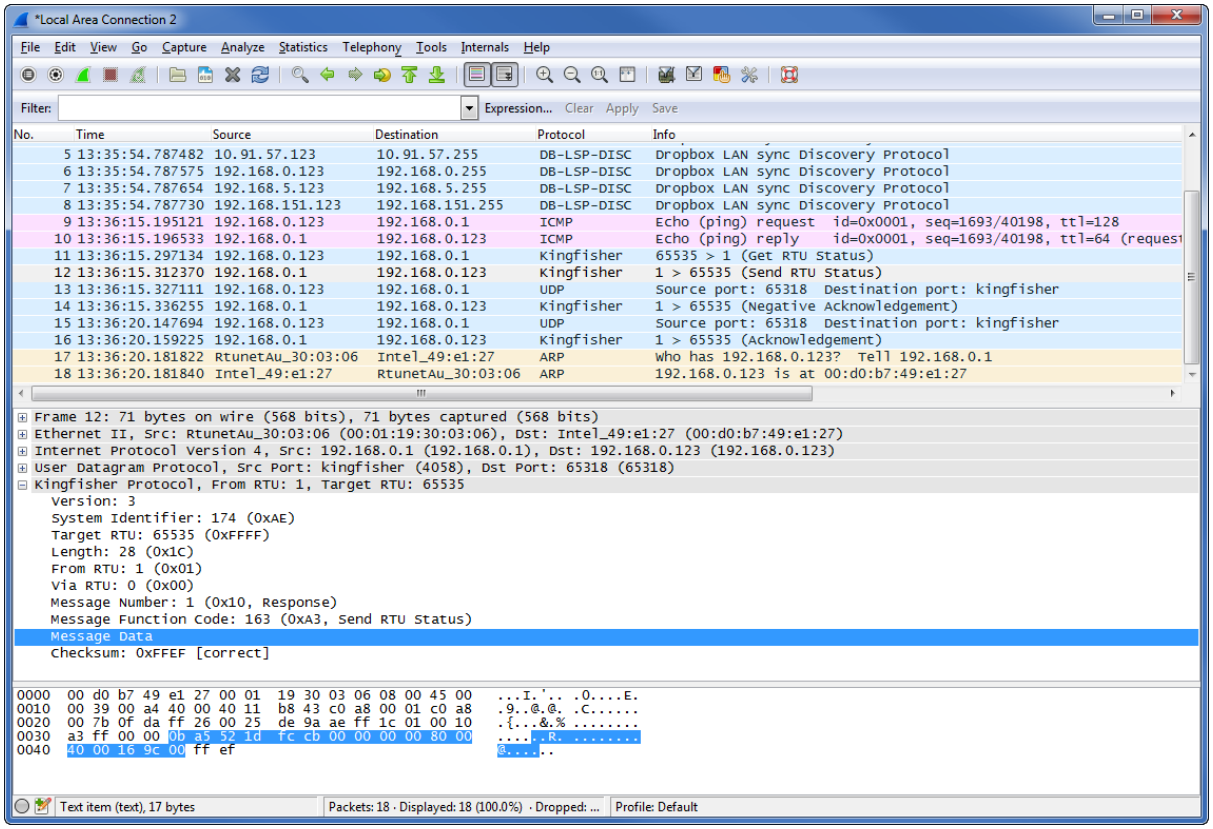
In the bottom diagram, RTU1 has been unplugged from the switch and plugged into a hub instead. The hub is also connected to the PC and to rest of the network via the switch. In this configuration anything sent to or from RTU1 (including traffic directed to RTU2) will be visible to the PC and will be able to be captured by Wireshark.

### *Using Wireshark*

Download Wireshark from www.wireshark.org and install on your PC, then start it as you would any application.

The Wireshark website includes extensive documentation and tutorials. In general terms, however the procedure is to first select the correct Ethernet interface (if your computer has more than one) from the list on the Wireshark home screen, then press *Start*.

A typical Wireshark display is shown below:


*Local Area Connection 2
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
No. Time Source Destination Protocol Info
5 13:35:54.787482 10.91.57.123 10.91.57.255 DB-LSP-DISC Dropbox LAN sync Discovery Protocol
6 13:35:54.787575 192.168.0.123 192.168.0.255 DB-LSP-DISC Dropbox LAN sync Discovery Protocol
7 13:35:54.787654 192.168.5.123 192.168.5.255 DB-LSP-DISC Dropbox LAN sync Discovery Protocol
8 13:35:54.787730 192.168.151.123 192.168.151.255 DB-LSP-DISC Dropbox LAN sync Discovery Protocol
9 13:36:15.195121 192.168.0.123 192.168.0.1 ICMP Echo (ping) request id=0x0001, seq=1693/40198, ttl=128
10 13:36:15.196533 192.168.0.1 192.168.0.123 ICMP Echo (ping) reply id=0x0001, seq=1693/40198, ttl=64 (reques
11 13:36:15.297134 192.168.0.123 192.168.0.1 Kingfisher 65535 > 1 (Get RTU Status)
12 13:36:15.312370 192.168.0.1 192.168.0.123 Kingfisher 1 > 65535 (Send RTU Status)
13 13:36:15.327111 192.168.0.123 192.168.0.1 UDP Source port: 65318 Destination port: kingfisher
14 13:36:15.336255 192.168.0.1 192.168.0.123 Kingfisher 1 > 65535 (Negative Acknowledgement)
15 13:36:20.147694 192.168.0.123 192.168.0.1 UDP Source port: 65318 Destination port: kingfisher
16 13:36:20.159225 192.168.0.1 192.168.0.123 Kingfisher 1 > 65535 (Acknowledgement)
17 13:36:20.181822 RtunetAu_30:03:06 Intel_49:e1:27 ARP Who has 192.168.0.123? Tell 192.168.0.1
18 13:36:20.181840 Intel_49:e1:27 RtunetAu_30:03:06 ARP 192.168.0.123 is at 00:d0:b7:49:e1:27
Frame 12: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
Ethernet II, Src: RtunetAu_30:03:06 (00:01:19:30:03:06), Dst: Intel_49:e1:27 (00:d0:b7:49:e1:27)
Internet Protocol Version 4, Src: 192.168.0.1 (192.168.0.1), Dst: 192.168.0.123 (192.168.0.123)
User Datagram Protocol, Src Port: kingfisher (4058), Dst Port: 65318 (65318)
Kingfisher Protocol, From RTU: 1, Target RTU: 65535
Version: 3
System Identifier: 174 (0xAE)
Target RTU: 65535 (0xFFFF)
Length: 28 (0x1C)
From RTU: 1 (0x01)
Via RTU: 0 (0x00)
Message Number: 1 (0x10, Response)
Message Function Code: 163 (0xA3, Send RTU Status)
Message Data
Checksum: 0xFFEF [correct]
0000 00 d0 b7 49 e1 27 00 01 19 30 03 06 08 00 45 00 ...I.'.. .0....E.
0010 00 39 00 a4 40 00 40 11 b8 43 c0 a8 00 01 c0 a8 .9..@.@. .C......
0020 00 7b 0f da ff 26 00 25 de 9a ae ff 1c 01 00 10 .{...&.% ........
0030 a3 ff 00 00 0b a5 52 1d fc cb 00 00 00 00 80 00 ....R. ........
0040 40 00 16 9c 00 ff ef @.....
Text item (text), 17 bytes
Packets: 18 · Displayed: 18 (100.0%) · Dropped: ...
Profile: Default

This shows, from top to bottom:

- a list of all received or transmitted packets
- details of the selected packet, with each protocol layer decoded (if possible)
- the raw data in the packet.

# 13. Appendices

## 13.1 Glossary

| | |
|---|---|
| **BYTE** | A group of 8 bits. Each bit can be a 0 (off) or a 1 (on) allowing up to 256 combinations. |
| **CP-30** | Kingfisher PLUS+ processor module. Runs the Linux operating system and supports ISaGRAF. |
| **DICTIONARY** | Set of variables available for use in logic programs or to be polled by a remote RTU. |
| **EXCEPTION REPORT** | A sporadic message initiated by a slave RTU to another RTU (usually to the master) to report new data or a significant event. |
| **FUNCTION BLOCK** | This term refers to a block of code that can be executed by an ISaGRAF logic program. Examples of function blocks include AGA gas calculations, hardware configuration and communication messages. |
| **G30-SA** | Kingfisher PLUS+ processor module. Runs the Linux operating system and supports ISaGRAF. Housed in a stand-alone (SA) enclosure and has one power supply card and one IO card (optional). |
| **INPUT** | Signal into the RTU |
| **IO** | Input, Output |
| **MASTER** | The device that is responsible for the collection, concentration and ultimate reporting of information from local and remote devices (conversely described as “slave” devices). The Master device may be an RTU or a PC running SCADA software.<br><br>The master device is usually responsible for initiating communications with slave devices (polling) but may also accept unsolicited messages from local and remote devices. |
| **OUTPUT** | Signal out of the RTU |
| **POLL** | A periodic message initiated by the master RTU to get the latest data and check the state of communications to a remote RTU. |
| **PORT** | A physical connection or socket on an RTU used for communications |
| **PROTOCOL** | Refers to the format of messages that may be passed to, from and through an RTU in communication with local and remote devices. Communications may use one or more RTU ports. Examples of protocols used within telemetry include Kingfisher, Modbus and DNP3. |
| **PS-1x** | Kingfisher PLUS+ power supply |
| **RTU** | Acronym for Remote Terminal Unit. Describes a group of processor, communication and IO modules that comprise a device for monitoring and control of hardware and devices in remote locations. |
| **SLAVE** | The device that is responsible for the collection of IO and other information. This data can then be polled or exception reported to a master device. A slave may be an RTU or another device. |
| **TCP/IP** | Transfer Control Protocol / Internet Protocol. Commonly used for Ethernet communications. |
| **UDP** | User Datagram Protocol. |

## 13.2 Kingfisher PLUS+ Specifications

The most significant difference between Kingfisher PLUS+ processor modules (PC-1, CP-11, CP-21, LP-x) is that the CP-30 and G30 processors support ISaGRAF while the other processors do not. To support ISaGRAF, CP-30 and G30 processors handle data in different ways as detailed below.

| Item | Non-ISaGRAF Kingfisher Processors | ISaGRAF Kingfisher Processors |
|---|---|---|
| **Processors** | PC-1, CP-11, CP-21, LP-x | CP-30 or G30-SA (stand alone) |
| **Data Limit** | 2048 Local or Network Registers | Virtually Unlimited |
| **Data Format** | 16-bit registers<br>Pairs of local registers are used to create long and float registers (32-bit)<br>Most registers are read/write | Variables of various types (Binary, Integer, Long Integer, Float etc) can be created in the Dictionary |
| **Storage Location** | All registers are stored in Static RAM.<br>Local registers are stored in a fixed location in memory.<br>Memory for Network registers is allocated dynamically as needed. | Memory is dynamically allocated to variables.<br>Data is stored in the variable itself.<br>Stores data from Non-ISaGRAF processors if the corresponding variables exist in the Dictionary. E.g. Creating the variable KF17N210 will allow the RTU to store local register #R210 from RTU17 |
| **Data Address** | The register address is a reference to where data is stored in RTU memory.<br>E.g. #R533, #DI14.1 | Variable names are used to address data. The RTU manages where the data is stored. A variable name is a label that describes the data it represents.<br>E.g. SL14IO4DI1, RTU23_ACPWR_STATUS |
| **RTU System Data** | System Registers (#Y…) allow access to RTU system data. | Toolbox PLUS+ automatically adds I/O module variables to the dictionary.<br>RTU System Data is available from custom function blocks in ISaGRAF. |
| **Available Data** | Registers do not have to be created. They are always available for use in ladder logic. | Data can only be accessed when variables are created in the Dictionary (except for IO module variables that are automatically created) |
| **Data Transfer** | Data transferred by referring to local or network registers | Data transferred by referring to variable names |

## 13.3 Kingfisher PLUS+ General

| Item | Non-ISaGRAF Kingfisher Processors | ISaGRAF Kingfisher Processors |
|---|---|---|
| **Configuration Software** | Toolbox 32 | Toolbox PLUS<br>ISaGRAF Workbench. All international IEC 61131 control languages are supported. |
| **Address Range** | 0-255 (8-bit) | 0-65535 (16-bit) |
| **Communications Path** | Called a network 'link'<br>Direct or indirect | Called a 'route'<br>Direct or Indirect |
| **Data Transfer** | Ladder Logic blocks | ISaGRAF Custom Function Blocks |
| **Port Referencing** | Port Number 1 to 16 | Module, Slot Number, Port 1 to 3 |
| **Memory Usage** | Must be manually allocated | Automatically allocated |
| **Drivers** | Download as required | Merged automatically by Toolbox PLUS |
| **Protocol Support** | Supported by some ports | Supported by all ports (provided hardware is suitable) |
| **Projects** | Optional. RTU configurations can be kept separate or included in a project. | A project must be created before an RTU configuration can be created. All RTU configurations are kept in a project. |
| **Kingfisher TCP/IP or UDP port number** | 473 | 473 |
| **Size of event logs** | 12 bytes | 40 bytes |
| **AI-10 input ranges** | One range for all 8 channels | One range per channel |

## 13.4 Kingfisher PLUS+ Protocol

ISaGRAF processors (CP-30, G30) are designed to be backwards compatible with non-ISaGRAF processors (PC-1, CP-11, CP-21, and LP-1).

ISaGRAF processors use 16-bit addresses (0-65535) while Non-ISaGRAF Processors use 8-bit addresses (0-255). If an RTU network with ISaGRAF processors only uses RTU addresses less than 256, then both types of processors are largely compatible for most applications.

An ISaGRAF processor can handle messages for addresses above 255 while non-ISaGRAF processor cannot.

| RTU Address | Description |
|---|---|
| **0** | All RTUs respond to address 0. |
| **1-249** | General addresses available to all RTUs |
| **250-255** | Reserved for Toolbox 32 / Toolbox PLUS+ and other PC applications |
| **256-65520** | Addresses available for ISaGRAF processor (CP-30, G30) RTUs |
| **65521-65535** | Reserved for Toolbox PLUS+ and other PC applications |

Detailed protocol information can be found in the manual - Kingfisher PLUS+ Protocol - available from the support website.

### 13.4.1 Kingfisher Message Structure

Each Kingfisher message has a maximum length of 255 bytes. ISaGRAF ™ processor modules use two bytes to store each RTU address while non-ISaGRAF ™ processor modules use one byte.

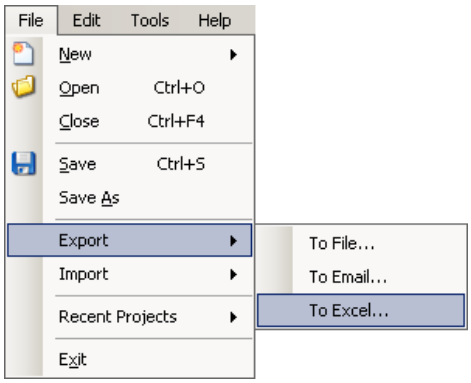
[SYSTEM ID][TARGET RTU][NO OF CHARACTERS][INITIATING RTU][VIA RTU][MESSAGE NO.][FUNCTION CODE] [DATA/ARGUMENTS][CRC]

| Message Component | Number of Bytes | | Description |
|---|---|---|---|
| | Non-ISaGRAF Processors | ISaGRAF Processors | |
| **SYSTEM ID** | 1 | 1 | Used to screen and identify incoming messages |
| **TARGET RTU** | 1 | 2 | Message destination |
| **NO OF CHARACTERS** | 1 | 1 | Characters in message excluding System ID and CRC |
| **INITIATING RTU** | 1 | 2 | Source of the message |
| **VIA RTU** | 1 | 2 | Relay RTU |
| **MESSAGE NO.** | 1 | 1 | Sequential Message number. Eg. FF: the first F indicates the message number. The second F is the reply number. If the second character is 0, then no reply. |
| **FUNCTION CODE** | 1 | 1 | Message type as per the protocol |
| **DATA/ARGUMENTS** | 246 max | 243 max | |
| **CRC** | 2 | 2 | Ensures data integrity |

## 13.5 Creating Variables Using Excel

Dictionary variables in an RTU configuration can be exported to a Microsoft Excel spreadsheet. Variables can then be edited and created using Excel. When the changes are complete, the spreadsheet is imported back into the RTU configuration, overwriting any existing variables. This method can be very useful for creating large quantities of variables with custom settings.

### 3.5.1 Exporting Variables

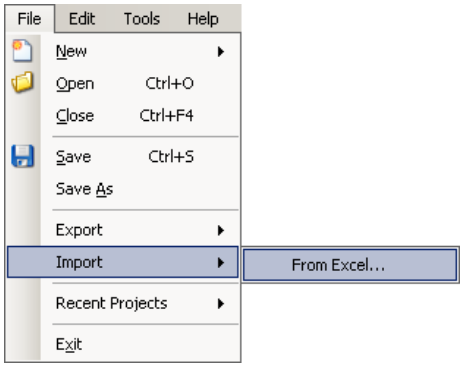



Select the RTU name in the navigation pane containing the variables to export

Select **File → Export → To Excel…**

Select a folder and filename to export the variables to

### 3.5.2 Importing Variables



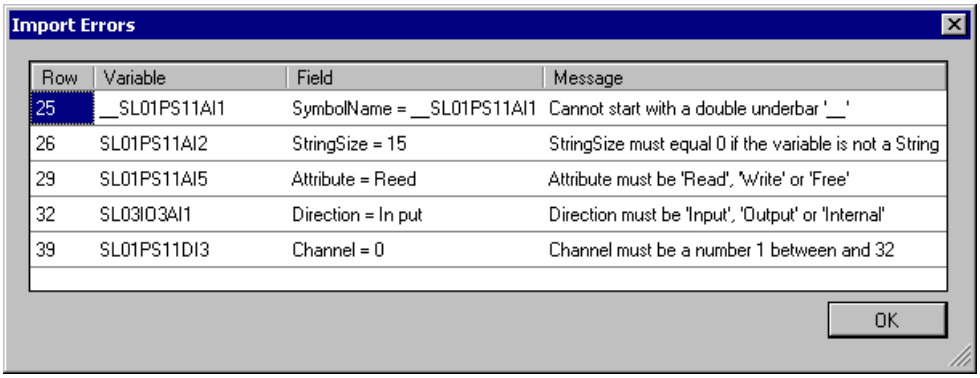

Select the RTU name in the navigation pane to store the imported variables in

Select **File, Import, From Excel…**

Select the Excel spreadsheet (<Filename>.xls) to import the variables from. Note: the spreadsheet must have the same columns as the spreadsheet created by the Export function above.

### 13.5.3 Importing Notes

- Before importing variables from an Excel spreadsheet into an RTU configuration, Toolbox PLUS+ extensively checks the spreadsheet for errors. If there are any errors, all variables will not be imported.
- When importing IO variables, the corresponding IO module must already exist in the RTU configuration. If an IO variable is assigned to a slot or channel that is not already defined in the RTU configuration, variables will not be imported.
- When errors are displayed, the row number in the spreadsheet is also displayed so that the error can be located. Examples:




Rows containing a blank **SymbolName** (variable name) are ignored

### 13.5.4 Spreadsheet Format

The spreadsheet is divided into four sections of columns as follows:

The **White** columns (A to M) contain basic variable parameters

| A | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SymbolName | Comment | SymbolType | Scope | StringSize | Attribute | Direction | Group | Retain | InitialValue | Array | Alias | Library |

The **Yellow** columns (N to Q) contain optional IO point parameters if the variable is to be linked to an IO Point

| N | O | P | Q |
|---|---|---|---|
| Slot | Card | IOType | Channel |

The **Blue** columns (R to AA) contain optional parameters for DNP3 variables. These parameters are only read if the variable has a valid name (**SymbolName**) and valid data type (SymbolType). For acceptable settings please see the DNP3 Implementation Guide available from the support website.

| R | S | T | U | V | W | X | Y | Z | AA |
|---|---|---|---|---|---|---|---|---|---|
| DnpClass | DnpFrozenClass | DnpStaticVariation | DnpFrozenStaticVariation | DnpEventVariation | DnpFrozenEventVariation | DnpDeadband | DnpHighLimit | DnpLowLimit | DnpRawLimits |

The **Green** column (AB) contains optional parameters for IEC variables. These parameters are only read if the variable has a valid name (**SymbolName**) and valid data type (**SymbolType**).

| AB |
|---|
| IECGroup |

Additional columns (AB onwards) can be used for comments and calculations as needed. Columns AB onwards will be ignored when the spreadsheet is imported back into Toolbox PLUS.

### 13.5.5 Spreadsheet Parameters

| Parameter | Description | Notes |
|---|---|---|
| **SymbolName** | The variable name | Maximum length of 128 characters<br>Only letters, numbers and underscore ( _ ) allowed<br>Cannot start with a double or single underscore<br>Cannot start with a number<br>Cannot be an ISaGRAF reserved name<br>Each variable must have a unique name |
| **Comment** | A comment for the variable | Maximum length of 128 characters |
| **SymbolType** | The data type | Must be a type defined and usable within ISaGRAF |
| **Scope** | Describes where this variable is available for use | The text must be either:<br>The word Global, which indicates that the variable can be used in all Functions, Function Blocks and Programs<br>The name of a specific Function, Function Block or Program that the Variable can only be used in |
| **StringSize** | The maximum number of characters in a String variable | If the variable is not a String, the value must be 0 |
| **Attribute** | A Read Only, Write Only or Read and Write variable | Must be: Read, Write or Free (read and write) |
| **Direction** | Describes the flow of IO data | If this is not attached to an IO Point, the value must be Internal. Otherwise, the value must be Input or Output |
| **Group** | The dictionary Group | Maximum length of 128 characters. If the Group does not already exist, a new Group will be created. |
| **Retain** | Retain variable value after a reboot | Set Retain to 1 to retain the variable value after a reboot of the RTU. Otherwise set to 0. |

| Parameter | Description | Notes |
|---|---|---|
| **InitialValue** | An initial value for the variable | The format of InitialValue depends on its SymbolType. Array values are entered as values separated by commas – example: 2, 3, 4. Strings are entered as text in single quotations – example: 'This is a string'<br>String arrays are entered as: 'first string','second string','third string'<br>This column is formatted as Text. Cells can be reformatted as Numeric to allow Excel calculations. To reformat, select cells or column J and then select the Excel menu – Format, Cells. From the Number tab, select Number. Once editing is complete, the cells or column must be re-formatted back to Text before the spreadsheet is imported into Toolbox PLUS. |
| **Array** | Array variables | Allows multidimensional arrays to be specified |
| **Alias** | | |
| **Library** | ISaGRAF Library | Allows reference to ISaGRAF library variables |
| **Slot** | The slot of the IO Point module | On a G30 RTU, Slot is always 1. If the variable is not attached to an IO Point, the value can be set to 1 or left blank. |
| **Card** | The number of the Option Card on the Module | If there is no Option Card, as is the case on CP-30 modules, the value is 1.<br>G30 MX cards are on option card 2.<br>G30 power supplies are on option card 3.<br>If the variable is not attached to an IO Point, the value can be set to either -1, or left blank. |
| **IOType** | The type of IO Point | Must be: AI, AO, DI, or DO |
| **Channel** | The IO channel number | If the variable is not attached to an IO Point, the value can be set to -1 or left blank |
| **DnpClass** | The Class of the DNP variable | Must be 0, 1, 2 or 3 |
| **DnpFrozenClass** | Frozen Class | Must be 0, 1, 2 or 3 |

<table>
<tr><th>Parameter</th><th>Description</th><th colspan="2">Notes</th></tr>
<tr><td rowspan="9">DnpStaticVariation</td><td rowspan="9">Static Variation</td><td colspan="2">Acceptable values depend on the IO Type of the variable as follows:</td></tr>
<tr><td>IO Type</td><td>Static Variation</td></tr>
<tr><td>Binary Input (BI)</td><td>1, 2</td></tr>
<tr><td>Binary Output (BO)</td><td>1, 2</td></tr>
<tr><td>Binary Counter (BC)</td><td>1, 2, 5, 6</td></tr>
<tr><td>Analog Input (AI), IOPOINT_D</td><td>1, 2, 3, 4</td></tr>
<tr><td>Analog Input (AI), IOPOINT_R</td><td>5</td></tr>
<tr><td>Analog Output (AO), IOPOINT_D</td><td>1, 2</td></tr>
<tr><td>Analog Output (AO), IOPOINT_R</td><td>3</td></tr>
<tr><td>DnpFrozenStaticVariation</td><td>Frozen Static Variation</td><td colspan="2">Only used by Binary Counter (BC) IO Types. Must be 1, 2, 5, 6, 9, or 10. All other IO Types ignore this column.</td></tr>
<tr><td rowspan="9">DnpEventVariation</td><td rowspan="9">Event Variation</td><td colspan="2">Acceptable values depend on the IO Type of the variable as follows:</td></tr>
<tr><td>IO Type</td><td>Event Variation</td></tr>
<tr><td>Binary Input (BI)</td><td>1, 2, 3</td></tr>
<tr><td>Binary Output (BO)</td><td>1, 2</td></tr>
<tr><td>Binary Counter (BC)</td><td>1, 2, 5, 6</td></tr>
<tr><td>Analog Input (AI), IOPOINT_D</td><td>1, 2, 3, 4</td></tr>
<tr><td>Analog Input (AI), IOPOINT_R</td><td>5, 7</td></tr>
<tr><td>Analog Output (AO), IOPOINT_D</td><td>1, 2, 3, 4</td></tr>
<tr><td>Analog Output (AO), IOPOINT_R</td><td>5, 7</td></tr>
<tr><td>DnpFrozenEventVariation</td><td>Frozen Event Variation</td><td colspan="2">Only used by Binary Counter (BC) IO Types. Must be 1, 2, 5, or 6. All other IO Types ignore this column.</td></tr>
<tr><td>DnpDeadband</td><td>The change in value required to trigger an Event</td><td colspan="2">Either a percentage value or a raw value depending on setting of DnpRawLimits</td></tr>
</table>

| Parameter | Description | Notes |
|---|---|---|
| **DnpHighLimit** | Highest value allowed for this variable | Higher settings are ignored and the value is limited to this set-point. Either a percentage value or a raw value can be specified depending on the setting of DnpRawLimits. |
| **DnpLowLimit** | Lowest value allowed for this variable | Lower settings are ignored and the value is limited to this set-point. Either a percentage value or a raw value can be specified depending on the setting of DnpRawLimits. |
| **DnpRawLimits** | Percentage or Raw values | Type of values used for DnpDeadband, DnpHighLimit, and DnpLowLimit. 0 = Percentage, 1 = Raw. |
| **IecGroup** | IEC variable group | An integer with range 0 to 16. A value of 0 represents a General group. |

## 13.6 RTU Variables

All data in Kingfisher PLUS+ RTUs are stored as named **variables**. Space is automatically allocated in RTU memory according to the number of variables that are created and the size of each variable. The set of defined variables is called the **dictionary**.

ISaGRAF programs can use any variables that have been created in the dictionary.

Variables fall into one of three categories:

- Variables which are automatically created when an I/O module is added to the RTU. The following sections describe the variables which are created for each of the various I/O module types.
- Manually created variables which follow a particular naming convention. These variables can be transferred to and from other systems using particular communications protocols.
- Manually created user variables, for holding temporary values and performing calculations.

### 13.6.1 Kingfisher PLUS+ I/O Modules

When an I/O module is added to an RTU, variables are automatically created in the dictionary that correspond to each input, output and data register (eg. counter) available from that module.

These variables are named: **SL***ssmmmmttnn*, where:

- **SL** is a fixed prefix (“slot”)
- *ss* is the slot number (01-64)
- *mmmm* is the module type, e.g. AI1, DI10
- *tt* is the type of I/O point, e.g. AI for analog inputs, DO for digital outputs
- *nn* is the I/O point number (1, 2, 3…)

The following sections list the automatically created variables for each I/O module type.

All I/O variables have either IOPOINT_B (for digital inputs/outputs) or IOPOINT_D (for analog inputs/outputs and counters) data type.

*AI-1 or AI-4*

8-channel analog input modules

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Input Ch1 | SLssAI1AI1 | R | Raw scale 0-32760 = 0-100% |
| Analog Input Ch2 | SLssAI1AI2 | R | |
| Analog Input Ch3 | SLssAI1AI3 | R | |
| Analog Input Ch4 | SLssAI1AI4 | R | |
| Analog Input Ch5 | SLssAI1AI5 | R | |
| Analog Input Ch6 | SLssAI1AI6 | R | |
| Analog Input Ch7 | SLssAI1AI7 | R | |
| Analog Input Ch8 | SLssAI1AI8 | R | |

### *AI-10*

8 channel bipolar analog input module with over range flags

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Input Ch1 | SLssAI10AI1 | R | Raw scale: -32768 to +32767 |
| Analog Input Ch2 | SLssAI10AI2 | R | |
| Analog Input Ch3 | SLssAI10AI3 | R | |
| Analog Input Ch4 | SLssAI10AI4 | R | |
| Analog Input Ch5 | SLssAI10AI5 | R | |
| Analog Input Ch6 | SLssAI10AI6 | R | |
| Analog Input Ch7 | SLssAI10AI7 | R | |
| Analog Input Ch8 | SLssAI10AI8 | R | |
| Ch1 Under Range Flag | SLssAI10DI1 | R | Set to TRUE when input current or voltage is below the minimum value. |
| Ch2 Under Range Flag | SLssAI10DI2 | R | |
| Ch3 Under Range Flag | SLssAI10DI3 | R | |
| Ch4 Under Range Flag | SLssAI10DI4 | R | |
| Ch5 Under Range Flag | SLssAI10DI5 | R | |
| Ch6 Under Range Flag | SLssAI10DI6 | R | |
| Ch7 Under Range Flag | SLssAI10DI7 | R | |
| Ch8 Under Range Flag | SLssAI10DI8 | R | |
| Ch1 Over Range Flag | SLssAI10DI9 | R | Set to TRUE when input current or voltage is above the maximum value. |
| Ch2 Over Range Flag | SLssAI10DI10 | R | |
| Ch3 Over Range Flag | SLssAI10DI11 | R | |
| Ch4 Over Range Flag | SLssAI10DI12 | R | |
| Ch5 Over Range Flag | SLssAI10DI13 | R | |
| Ch6 Over Range Flag | SLssAI10DI14 | R | |
| Ch7 Over Range Flag | SLssAI10DI15 | R | |
| Ch8 Over Range Flag | SLssAI10DI16 | R | |

### *AO-2*

4 channel analog output module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Output Ch1 | SLssAO2AO1 | R/W | Raw scale 0-32760 = 0-100% |
| Analog Output Ch2 | SLssAO2AO2 | R/W | |
| Analog Output Ch3 | SLssAO2AO3 | R/W | |
| Analog Output Ch4 | SLssAO2AO4 | R/W | |

### *AO-3*

4 channel analog output module with open loop detection

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Output Ch1 | SLssAO3AO1 | R/W | Raw scale 0-32760 = 0-100% |
| Analog Output Ch2 | SLssAO3AO2 | R/W | |
| Analog Output Ch3 | SLssAO3AO3 | R/W | |
| Analog Output Ch4 | SLssAO3AO4 | R/W | |
| Ch1 Open Loop | SLssAO3DI1 | R | 1 = Open Loop |
| Ch2 Open Loop | SLssAO3DI2 | R | |
| Ch3 Open Loop | SLssAO3DI3 | R | |
| Ch4 Open Loop | SLssAO3DI4 | R | |

### *DI-1*

16 channel digital input module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssDI1DI1 | R | |
| Digital Input Ch2 | SLssDI1DI2 | R | |
| Digital Input Ch3 | SLssDI1DI3 | R | |
| Digital Input Ch4 | SLssDI1DI4 | R | |
| Digital Input Ch5 | SLssDI1DI5 | R | |
| Digital Input Ch6 | SLssDI1DI6 | R | |
| Digital Input Ch7 | SLssDI1DI7 | R | |
| Digital Input Ch8 | SLssDI1DI8 | R | |
| Digital Input Ch9 | SLssDI1DI9 | R | |
| Digital Input Ch10 | SLssDI1DI10 | R | |
| Digital Input Ch11 | SLssDI1DI11 | R | |
| Digital Input Ch12 | SLssDI1DI12 | R | |
| Digital Input Ch13 | SLssDI1DI13 | R | |
| Digital Input Ch14 | SLssDI1DI14 | R | |
| Digital Input Ch15 | SLssDI1DI15 | R | |
| Digital Input Ch16 | SLssDI1DI16 | R | |

### *DI-5*

16 channel digital input / 4 counter module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssDI5DI1 | R | The first 4 channels interface to hardware counters.<br>Channels 1 and 2 up to 10kHz.<br>Channels 3 & 4 up to 255Hz.<br><br>If the maximum pulse rate is exceeded on channels 3 and 4 (>255 Hz), each counter will contain the lowest 8 bits of the actual pulse rate. Results are unpredictable at pulse rates greater that 1 kHz |
| Digital Input Ch2 | SLssDI5DI2 | R | |
| Digital Input Ch3 | SLssDI5DI3 | R | |
| Digital Input Ch4 | SLssDI5DI4 | R | |
| Digital Input Ch5 | SLssDI5DI5 | R | |
| Digital Input Ch6 | SLssDI5DI6 | R | |
| Digital Input Ch7 | SLssDI5DI7 | R | |
| Digital Input Ch8 | SLssDI5DI8 | R | |
| Digital Input Ch9 | SLssDI5DI9 | R | |
| Digital Input Ch10 | SLssDI5DI10 | R | |
| Digital Input Ch11 | SLssDI5DI11 | R | |
| Digital Input Ch12 | SLssDI5DI12 | R | |
| Digital Input Ch13 | SLssDI5DI13 | R | |
| Digital Input Ch14 | SLssDI5DI14 | R | |
| Digital Input Ch15 | SLssDI5DI15 | R | |
| Digital Input Ch16 | SLssDI5DI16 | R | |
| Ch1 Total Pulses | SLssDI5AI1 | R/W | |
| Ch2 Total Pulses | SLssDI5AI2 | R/W | |
| Ch3 Total Pulses | SLssDI5AI3 | R/W | |
| Ch4 Total Pulses | SLssDI5AI4 | R/W | |
| Ch1 Pulse Rate Hz | SLssDI5AI5 | R/W | |
| Ch2 Pulse Rate Hz | SLssDI5AI6 | R/W | |
| Ch3 Pulse Rate Hz | SLssDI5AI7 | R/W | |
| Ch4 Pulse Rate Hz | SLssDI5AI8 | R/W | |

### *DI-10*

16 channel digital input / 7 counter module with event time-stamping

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssDI10DI1 | R | Has configurable Frequency or Pulse or Quadrature counting and Sequence-of-Events recording (using event logs).<br><br>Counters 1-7 can count Frequency (0-10kHz max.) or Total Pulses (0-65535) or Quadrature Count (0-65535) for any configured channel(s) (as configured using Toolbox PLUS) |
| Digital Input Ch2 | SLssDI10DI2 | R | |
| Digital Input Ch3 | SLssDI10DI3 | R | |
| Digital Input Ch4 | SLssDI10DI4 | R | |
| Digital Input Ch5 | SLssDI10DI5 | R | |
| Digital Input Ch6 | SLssDI10DI6 | R | |
| Digital Input Ch7 | SLssDI10DI7 | R | |
| Digital Input Ch8 | SLssDI10DI8 | R | |
| Digital Input Ch9 | SLssDI10DI9 | R | |
| Digital Input Ch10 | SLssDI10DI10 | R | |
| Digital Input Ch11 | SLssDI10DI11 | R | |
| Digital Input Ch12 | SLssDI10DI12 | R | |
| Digital Input Ch13 | SLssDI10DI13 | R | |
| Digital Input Ch14 | SLssDI10DI14 | R | |
| Digital Input Ch15 | SLssDI10DI15 | R | |
| Digital Input Ch16 | SLssDI10DI16 | R | |
| Counter 1 | SLssDI10AI1 | R/W | |
| Counter 2 | SLssDI10AI2 | R/W | |
| Counter 3 | SLssDI10AI3 | R/W | |
| Counter 4 | SLssDI10AI4 | R/W | |
| Counter 5 | SLssDI10AI5 | R/W | |
| Counter 6 | SLssDI10AI6 | R/W | |
| Counter 7 | SLssDI10AI7 | R/W | |

### *DO-1*

8 channel digital output module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Output Ch1 | SLssDO1DO1 | R/W | |
| Digital Output Ch2 | SLssDO1DO2 | R/W | |
| Digital Output Ch3 | SLssDO1DO3 | R/W | |
| Digital Output Ch4 | SLssDO1DO4 | R/W | |
| Digital Output Ch5 | SLssDO1DO5 | R/W | |
| Digital Output Ch6 | SLssDO1DO6 | R/W | |
| Digital Output Ch7 | SLssDO1DO7 | R/W | |
| Digital Output Ch8 | SLssDO1DO8 | R/W | |

### *DO-2/ DO-5 / DO-6*

16 channel digital output modules

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Output Ch1 | SLssDO2DO1 | R/W | DO-2 has relays in the module<br>DO-5 and DO-6 have transistor outputs for driving an external relay board. |
| Digital Output Ch2 | SLssDO2DO2 | R/W | |
| Digital Output Ch3 | SLssDO2DO3 | R/W | |
| Digital Output Ch4 | SLssDO2DO4 | R/W | |
| Digital Output Ch5 | SLssDO2DO5 | R/W | |
| Digital Output Ch6 | SLssDO2DO6 | R/W | |
| Digital Output Ch7 | SLssDO2DO7 | R/W | |
| Digital Output Ch8 | SLssDO2DO8 | R/W | |
| Digital Output Ch9 | SLssDO2DO9 | R/W | |
| Digital Output Ch10 | SLssDO2DO10 | R/W | |
| Digital Output Ch11 | SLssDO2DO11 | R/W | |
| Digital Output Ch12 | SLssDO2DO12 | R/W | |
| Digital Output Ch13 | SLssDO2DO13 | R/W | |
| Digital Output Ch14 | SLssDO2DO14 | R/W | |
| Digital Output Ch15 | SLssDO2DO15 | R/W | |
| Digital Output Ch16 | SLssDO2DO16 | R/W | |

### *IO-2*

8 digital input / 8 digital output module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssIO2DI1 | R | |
| Digital Input Ch2 | SLssIO2DI2 | R | |
| Digital Input Ch3 | SLssIO2DI3 | R | |
| Digital Input Ch4 | SLssIO2DI4 | R | |
| Digital Input Ch5 | SLssIO2DI5 | R | |
| Digital Input Ch6 | SLssIO2DI6 | R | |
| Digital Input Ch7 | SLssIO2DI7 | R | |
| Digital Input Ch8 | SLssIO2DI8 | R | |
| Digital Output Ch1 | SLssIO2DO1 | R/W | |
| Digital Output Ch2 | SLssIO2DO2 | R/W | |
| Digital Output Ch3 | SLssIO2DO3 | R/W | |
| Digital Output Ch4 | SLssIO2DO4 | R/W | |
| Digital Output Ch5 | SLssIO2DO5 | R/W | |
| Digital Output Ch6 | SLssIO2DO6 | R/W | |
| Digital Output Ch7 | SLssIO2DO7 | R/W | |
| Digital Output Ch8 | SLssIO2DO8 | R/W | |

### *IO-3*

4 digital input / 4 digital output / 4 analog input / 1 analog output module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssIO3DI1 | R | Analog raw scale 0-32760 = 0-100% |
| Digital Input Ch2 | SLssIO3DI2 | R | |
| Digital Input Ch3 | SLssIO3DI3 | R | |
| Digital Input Ch4 | SLssIO3DI4 | R | |
| Digital Output Ch1 | SLssIO3DO1 | R/W | |
| Digital Output Ch2 | SLssIO3DO2 | R/W | |
| Digital Output Ch3 | SLssIO3DO3 | R/W | |
| Digital Output Ch4 | SLssIO3DO4 | R/W | |
| Analog Input Ch1 | SLssIO3AI1 | R | |
| Analog Input Ch2 | SLssIO3AI2 | R | |
| Analog Input Ch3 | SLssIO3AI3 | R | |
| Analog Input Ch4 | SLssIO3AI4 | R | |
| Analog Output Ch1 | SLssIO3AO1 | R/W | |

### *IO-4*

8 digital input / 2 digital output / 2 analog input module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssIO4DI1 | R | Analog raw scale 0-32760 = 0-100% |
| Digital Input Ch2 | SLssIO4DI2 | R | |
| Digital Input Ch3 | SLssIO4DI3 | R | |
| Digital Input Ch4 | SLssIO4DI4 | R | |
| Digital Input Ch5 | SLssIO4DI5 | R | |
| Digital Input Ch6 | SLssIO4DI6 | R | |
| Digital Input Ch7 | SLssIO4DI7 | R | |
| Digital Input Ch8 | SLssIO4DI8 | R | |
| Digital Output Ch1 | SLssIO4DO1 | R/W | |
| Digital Output Ch2 | SLssIO4DO2 | R/W | |
| Analog Input Ch1 | SLssIO4AI1 | R | |
| Analog Input Ch2 | SLssIO4AI2 | R | |

*IO-5*

4 digital input / 4 digital output / 4 analog input / 4 counter module

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SLssIO5DI1 | R | |
| Digital Input Ch2 | SLssIO5DI2 | R | |
| Digital Input Ch3 | SLssIO5DI3 | R | |
| Digital Input Ch4 | SLssIO5DI4 | R | |
| Digital Output Ch1 | SLssIO5DO1 | R/W | |
| Digital Output Ch2 | SLssIO5DO2 | R/W | |
| Digital Output Ch3 | SLssIO5DO3 | R/W | |
| Digital Output Ch4 | SLssIO5DO4 | R/W | |
| Analog Input Ch1 | SLssIO5AI1 | R | Analog raw scale 0-32760 = 0-100% |
| Analog Input Ch2 | SLssIO5AI2 | R | |
| Analog Input Ch3 | SLssIO5AI3 | R | |
| Analog Input Ch4 | SLssIO5AI4 | R | |
| Digital Input Counter 1 | SLssIO5AI5 | R | |
| Digital Input Counter 2 | SLssIO5AI6 | R | |
| Digital Input Counter 3 | SLssIO5AI7 | R | |
| Digital Input Counter 4 | SLssIO5AI8 | R | |

### *PS-1x or PS-2x*

Power supply modules

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Supply Voltage | SLssPS11AI1 | R | Raw Scale= 0-32736, Eng. Units=0 to 32.27V<br>The DC voltage supplied to the RTU modules on the backplane (typically 12V) and used to charge the battery. This voltage is sourced from the battery if there is no input supply present. |
| Battery Current | SLssPS11AI2 | R | Raw Scale=0-32736, Eng. Units= -4 to +4 A<br>Current is positive when charging. Negative when battery is discharging. |
| Total Current | SLssPS11AI3 | R | Raw Scale=0-32736, Eng. Units= -4 to +4 A<br>Total current supplied by the power supply to the RTU modules and battery. |
| Battery Temperature | SLssPS11AI4 | R | Raw Scale=0-32736<br>Eng. Units= -20 to +80 °C, 0 °C = 6547 |
| Battery Type | SLssPS11AI5 | R | 0 = Default, 1 = Lead-Acid, 2 = Ni-Cad |
| Battery Size | SLssPS11AI6 | R | Battery Size (x 0.1AH) 0 to 250 = 0 to 25.0 AH Max. |
| Module Temperature | SLssPS11AI7 | R | Raw Scale=0-32736,<br>Eng. Units= -20 to +80 °C, 0 °C = 6547 |
| Power ON | SLssPS11DI1 | R | 1 = AC (PS-1x) or DC (PS-2x) Power ON |
| AUX 24V Fail | SLssPS11DI2 | R | 1 = Auxiliary 24V failure or not present |
| Battery Low | SLssPS11DI3 | R | 0 = Battery low. Note: Active low.<br>This bit does not indicate if a battery is present as Battery Low is cleared whenever the input supply is active. If the input supply is OFF (ie. SLssPS11DI1=0), a battery is present if the RTU is still running! |
| Power Supply Type | SLssPS11DI4 | R | 1 = PS-2x, 0 = PS-1x |
| Float State | SLssPS11DI5 | R | 1 = float state |
| Charge State | SLssPS11DI6 | R | 1 = charge state |
| Boost State | SLssPS11DI7 | R | 1 = boost state |
| Temperature Sensor Error | SLssPS11DI8 | R | 1 = sensor error |
| Manual Power Control | SLssPS11DO1 | R/W | 0 = automatic control (default)<br>1 = manual control. Allows manual control of Radio and 24V power (and Mains Supply for PS-1x) |
| Radio power OFF | SLssPS11DO2 | R/W | 1 = radio OFF (if SLssPS11DO1=1) |
| Aux 24V OFF | SLssPS11DO3 | R/W | 1 = 24V OFF (if SLssPS11DO1=1) |
| Inhibit AC Supply Input Circuit (PS-1x only) | SLssPS11DO4 | R/W | 1 = inhibit AC (if SLssPS11DO1=1) |

### 13.6.2 G30 I/O Cards

When an IO or power supply card is added to a G30 RTU, variables are automatically created in the dictionary that correspond to each input, output and data register (eg. counter) available from that card.

These variables are named: **SL01***mmmmttnn*, where:

- **SL01** is a fixed prefix (“slot 1”)
- *mmmm* is the card type, e.g. MX2
- *tt* is the type of I/O point, e.g. AI for analog inputs, DO for digital outputs
- *nn* is the I/O point number (1, 2, 3…)

The following sections list the automatically created variables for each I/O card type.

*IOD-MX2*

14 digital input / 14 counter / 8 digital output / 6 analog input card

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Input Ch1 | SL01MX2AI1 | R | Analog raw scale 0-65534 = 0-100% |
| Analog Input Ch2 | SL01MX2AI2 | R | |
| Analog Input Ch3 | SL01MX2AI3 | R | |
| Analog Input Ch4 | SL01MX2AI4 | R | |
| Analog Input Ch5 | SL01MX2AI5 | R | |
| Analog Input Ch6 | SL01MX2AI6 | R | |
| Digital Input Ch1 | SL01MX2DI1 | R | Each input channel can be configured to measure Frequency or count pulses. When the digital input mode is changed to Frequency or Pulse, the channel’s variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 8: 10 kHz maximum<br>Channels 9 to 14: 500 Hz maximum<br><br>The following pairs of channels can also be configured as quadrature counters: 3&4, 5&6, 7&8, 9&10, 11&12, 13&14. When Mode is changed to Quad Counter for the odd-numbered channel, the even number channel variable is removed from the Dictionary, and the odd-numbered channel’s type is changed to IOPOINT_D (double integer). |
| Digital Input Ch2 | SL01MX2DI2 | R | |
| Digital Input Ch3 | SL01MX2DI3 | R | |
| Digital Input Ch4 | SL01MX2DI4 | R | |
| Digital Input Ch5 | SL01MX2DI5 | R | |
| Digital Input Ch6 | SL01MX2DI6 | R | |
| Digital Input Ch7 | SL01MX2DI7 | R | |
| Digital Input Ch8 | SL01MX2DI8 | R | |
| Digital Input Ch9 | SL01MX2DI9 | R | |
| Digital Input Ch10 | SL01MX2DI10 | R | |
| Digital Input Ch11 | SL01MX2DI11 | R | |
| Digital Input Ch12 | SL01MX2DI12 | R | |
| Digital Input Ch13 | SL01MX2DI13 | R | |
| Digital Input Ch14 | SL01MX2DI14 | R | |
| Digital Output Ch1 | SL01MX2DO1 | R/W | Each output channel can be configured to output Frequency or pulses. When the digital output mode is changed to Frequency or Pulse, the channel’s variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 4: 10 kHz maximum<br>Channels 5 to 8: 500 Hz maximum |
| Digital Output Ch2 | SL01MX2DO2 | R/W | |
| Digital Output Ch3 | SL01MX2DO3 | R/W | |
| Digital Output Ch4 | SL01MX2DO4 | R/W | |
| Digital Output Ch5 | SL01MX2DO5 | R/W | |
| Digital Output Ch6 | SL01MX2DO6 | R/W | |
| Digital Output Ch7 | SL01MX2DO7 | R/W | |
| Digital Output Ch8 | SL01MX2DO8 | R/W | |

*IOD-MX3*

14 digital input / 14 counter / 8 digital output / 6 analog input / 2 analog output card

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Analog Input Ch1 | SL01MX3AI1 | R | Analog raw scale 0-65534 = 0-100% |
| Analog Input Ch2 | SL01MX3AI2 | R | |
| Analog Input Ch3 | SL01MX3AI3 | R | |
| Analog Input Ch4 | SL01MX3AI4 | R | |
| Analog Input Ch5 | SL01MX3AI5 | R | |
| Analog Input Ch6 | SL01MX3AI6 | R | |
| Analog Output Ch1 | SL01MX3AO1 | R/W | Analog raw scale 0-65534 = 0-100% |
| Analog Output Ch2 | SL01MX3AO2 | R/W | |
| Digital Input Ch1 | SL01MX3DI1 | R | Each input channel can be configured to measure Frequency or count pulses. When the digital input mode is changed to Frequency or Pulse, the channel's variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 8: 10 kHz maximum<br>Channels 9 to 14: 500 Hz maximum<br><br>The following pairs of channels can also be configured as quadrature counters: 3&4, 5&6, 7&8, 9&10, 11&12, 13&14. When Mode is changed to Quad Counter for the odd-numbered channel, the even number channel variable is removed from the Dictionary, and the odd-numbered channel's type is changed to IOPOINT_D (double integer). |
| Digital Input Ch2 | SL01MX3DI2 | R | |
| Digital Input Ch3 | SL01MX3DI3 | R | |
| Digital Input Ch4 | SL01MX3DI4 | R | |
| Digital Input Ch5 | SL01MX3DI5 | R | |
| Digital Input Ch6 | SL01MX3DI6 | R | |
| Digital Input Ch7 | SL01MX3DI7 | R | |
| Digital Input Ch8 | SL01MX3DI8 | R | |
| Digital Input Ch9 | SL01MX3DI9 | R | |
| Digital Input Ch10 | SL01MX3DI10 | R | |
| Digital Input Ch11 | SL01MX3DI11 | R | |
| Digital Input Ch12 | SL01MX3DI12 | R | |
| Digital Input Ch13 | SL01MX3DI13 | R | |
| Digital Input Ch14 | SL01MX3DI14 | R | |
| Digital Output Ch1 | SL01MX3DO1 | R/W | Each output channel can be configured to output Frequency or pulses. When the digital output mode is changed to Frequency or Pulse, the channel's variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 4: 10 kHz maximum<br>Channels 5 to 8: 500 Hz maximum |
| Digital Output Ch2 | SL01MX3DO2 | R/W | |
| Digital Output Ch3 | SL01MX3DO3 | R/W | |
| Digital Output Ch4 | SL01MX3DO4 | R/W | |
| Digital Output Ch5 | SL01MX3DO5 | R/W | |
| Digital Output Ch6 | SL01MX3DO6 | R/W | |
| Digital Output Ch7 | SL01MX3DO7 | R/W | |
| Digital Output Ch8 | SL01MX3DO8 | R/W | |

*IOD-MX4*

16 digital input / 16 counter / 16 digital output card

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Digital Input Ch1 | SL01MX4DI1 | R | Each input channel can be configured to measure Frequency or count pulses. When the digital input mode is changed to Frequency or Pulse, the channel's variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 8: 10 kHz maximum<br>Channels 9 to 16: 500 Hz maximum<br><br>The following pairs of channels can also be configured as quadrature counters: 3&4, 5&6, 7&8, 9&10, 11&12, 13&14, 15&16. When Mode is changed to Quad Counter for the odd-numbered channel, the even number channel variable is removed from the Dictionary, and the odd-numbered channel's type is changed to IOPOINT_D (double integer). |
| Digital Input Ch2 | SL01MX4DI2 | R | |
| Digital Input Ch3 | SL01MX4DI3 | R | |
| Digital Input Ch4 | SL01MX4DI4 | R | |
| Digital Input Ch5 | SL01MX4DI5 | R | |
| Digital Input Ch6 | SL01MX4DI6 | R | |
| Digital Input Ch7 | SL01MX4DI7 | R | |
| Digital Input Ch8 | SL01MX4DI8 | R | |
| Digital Input Ch9 | SL01MX4DI9 | R | |
| Digital Input Ch10 | SL01MX4DI10 | R | |
| Digital Input Ch11 | SL01MX4DI11 | R | |
| Digital Input Ch12 | SL01MX4DI12 | R | |
| Digital Input Ch13 | SL01MX4DI13 | R | |
| Digital Input Ch14 | SL01MX4DI14 | R | |
| Digital Input Ch15 | SL01MX4DI15 | R | |
| Digital Input Ch16 | SL01MX4DI16 | R | |
| Digital Output Ch1 | SL01MX4DO1 | R/W | Each output channel can be configured to output Frequency or pulses. When the digital output mode is changed to Frequency or Pulse, the channel's variable type is changed to IOPOINT_R (real) or IOPOINT_D (double integer) respectively.<br><br>Channels 1 to 4: 10 kHz maximum<br>Channels 5 to 16: 500 Hz maximum |
| Digital Output Ch2 | SL01MX4DO2 | R/W | |
| Digital Output Ch3 | SL01MX4DO3 | R/W | |
| Digital Output Ch4 | SL01MX4DO4 | R/W | |
| Digital Output Ch5 | SL01MX4DO5 | R/W | |
| Digital Output Ch6 | SL01MX4DO6 | R/W | |
| Digital Output Ch7 | SL01MX4DO7 | R/W | |
| Digital Output Ch8 | SL01MX4DO8 | R/W | |
| Digital Output Ch9 | SL01MX4DO9 | R/W | |
| Digital Output Ch10 | SL01MX4DO10 | R/W | |
| Digital Output Ch11 | SL01MX4DO11 | R/W | |
| Digital Output Ch12 | SL01MX4DO12 | R/W | |
| Digital Output Ch13 | SL01MX4DO13 | R/W | |
| Digital Output Ch14 | SL01MX4DO14 | R/W | |
| Digital Output Ch15 | SL01MX4DO15 | R/W | |
| Digital Output Ch16 | SL01MX4DO16 | R/W | |

### *PSO-ACR*

AC input power supply card

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Regulator Voltage | SL01ACRAI1 | R | 12V regulator output voltage, in mV (0 to 32000) |
| Supply Current | SL01ACRAI2 | R | Current being drawn from regulator, in mA (-4000 to 4000) |
| Battery Current | SL01ACRAI3 | R | Battery Current (positive=charging), in mA (-4000 to 4000) |
| Internal Temp | SL01ACRAI4 | R | PCB Temperature in units of 0.01 °C (-2000 to 8000) |
| External Temp | SL01ACRAI5 | R | External sensor (LM335A) temperature, in units of 0.01 °C (-2000 to 8000) |
| - | SL01ACRAO1 | R/W | not used |
| DC Output Status | SL01ACRDI1 | R | Indicates main output supply status:<br>0: OFF/Overload, 1: ON/OK |
| Aux Output Status | SL01ACRDI2 | R | Indicates status of Aux supply:<br>0: OFF/Overload, 1: ON/OK |
| AC Input Statys | SL01ACRDI3 | R | Indicates the status of AC input power:<br>0: absent/no power, 1: present |
| Charging Status | SL01ACRDI4 | R | Indicates whether the battery is being charged or not<br>0: Not charging, 1: Charging |
| - | SL01ACRDO1 | R/W | not used |
| - | SL01ACRDO2 | R/W | not used |
| Aux Output Control | SL01ACRDO3 | R/W | Switches the auxiliary 12V output on or off<br>0: Output OFF, 1: Output ON |

### *PSO-DCU*

DC input power supply card

| Data Description | Variable Name | R/W | Notes |
|---|---|---|---|
| Supply Voltage | SL01DCUAI1 | R | External 12V supply input voltage, in mV (0 to 32000) |
| Supply Current | SL01DCUAI2 | R | Current drawn from external supply, in mA (0 to 4000) |
| - | SL01DCUAI3 | R | not used |
| Internal Temp | SL01DCUAI4 | R | PCB Temperature in units of 0.01 °C (-2000 to 8000) |
| External Temp | SL01DCUAI5 | R | External sensor (LM335A) temperature in units of 0.01 °C (-2000 to 8000) |
| - | SL01DCUAO1 | R/W | not used |
| DC Output Status | SL01DCUDI1 | R | Indicates main output supply status:<br>0: OFF/Overload, 1: ON/OK |
| Aux Output Status | SL01DCUDI2 | R | Indicates status of Aux supply:<br>0: OFF/Overload, 1: ON/OK |
| - | SL01ACRDI3 | R | not used |
| - | SL01ACRDI4 | R | not used |
| - | SL01ACRDO1 | R/W | not used |
| - | SL01ACRDO2 | R/W | not used |
| Aux Output Control | SL01ACRDO3 | R/W | Switches the auxiliary 12V output on or off<br>0: Output OFF, 1: Output ON |

### 13.6.3 Kingfisher Register Variables

Kingfisher local register variables are used to store 16-bit data values to be transferred between RTUs using the Kingfisher, Allen Bradley DF1 or HART protocols.

Variables must be manually created in the Dictionary before they can be used in logic or transferred via a protocol.

Kingfisher register variable names have the format: **KF***r***R***n*, where

- *r* is the address of the RTU from which the variable originated. If the variable belongs to the local RTU then this is blank.
- *n* is the register number (1-2048)

Floating point Kingfisher registers (type REAL) may also be created, which are named **KF***r***F***n*. These are only supported by the HART protocol, however.

In summary:

| Variable Name | Type | Description |
|---|---|---|
| KF*r*R*n* | DINT | Kingfisher integer register #*n* on RTU *r* |
| KF*r*F*n* | REAL | Kingfisher floating point register #*n* on RTU *r* |

For example, local Kingfisher register #315 would be named KFR315, while register #210 retrieved from RTU 17 would be KFR17R210.

### 13.6.4 Modbus Variables

Modbus variables are used to store 1-bit or 16-bit data values to be transferred between RTUs using the Modbus protocol.

Variables must be manually created in the Dictionary before they can be used in logic or transferred via Modbus.

Modbus variable names have the format: **MOD***r**T**n*, where

- *r* is the address of the RTU from which the variable originated. If the variable belongs to the local RTU then this is blank.
- *T* is the type of Modbus variable (see below)
- *n* is the register number (1-65535)

The following variable types may be created:

| Variable Name | Type | Description |
|---|---|---|
| MOD*r*C*n* | BOOL | Modbus coil #*n* on RTU *r* |
| MOD*r*D*n* | BOOL | Modbus discrete input #*n* on RTU *r* |
| MOD*r*H*n* | DINT | Modbus holding register #*n* on RTU *r* |
| MOD*r*I*n* | DINT | Modbus input register #*n* on RTU *r* |

For example, the variable for Modbus coil #22 on the local RTU would be named MODC22, while Modbus input register #2 read from RTU 39 would be MOD39I2.

Note that only the least significant byte of the RTU address (*r*) will be used in Modbus messages. This means that two RTUs whose addresses have the same least significant

byte (e.g. addresses 3 and 259) cannot be connected on the same network if Modbus is to be used.

Note also that some device documentation or SCADA systems identify the type of Modbus point by adding a numeric prefix, e.g. holding registers are in the range 40001-49999 or 400001-465535. The point number (*n*) specified in the variable name should not include this prefix digit, e.g. point 40006 would correspond to variable MODH6.

### 13.6.5 DNP3 Variables

DNP3 variables are used to store data values to be transferred between RTUs using the DNP3 protocol.

Variables must be manually created in the Dictionary before they can be used in logic or transferred via Modbus.

DNP3 variable names have the format: **DNP***rTn*, where

- *r* is the address of the RTU from which the variable originated. If the variable belongs to the local RTU then this is blank.
- *T* is the type of DNP3 variable (see below)
- *n* is the register number (0-65535)

The following variable types may be created:

| Variable Name | Type | Description |
|---|---|---|
| DNP*r*BI*n* | IOPOINT_B | Binary input #*n* on RTU *r* |
| DNP*r*BO*n* | IOPOINT_B | Binary output #*n* on RTU *r* |
| DNP*r*BC*n* | IOPOINT_D | Binary counter #*n* on RTU *r* |
| DNP*r*FC*n* | IOPOINT_D | Frozen counter #*n* on RTU *r* |
| DNP*r*AI*n* | IOPOINT_D or IOPOINT_R | Analog input #*n* on RTU *r* |
| DNP*r*AO*n* | IOPOINT_D or IOPOINT_R | Analog output #*n* on RTU *r* |

For example, the variable for DNP3 binary input #0 on the local RTU would be named DNPBI0, while DNP3 analog input #21 read from RTU 909 would be DNP909AI21.

DNP3 variables use the IOPOINT_*x* structure types, which include timestamp and flag information as well as the actual data value.

### 13.6.6 DNP3 Internal Indication Register

The DNP3 protocol defines a 16-bit Internal Indication (IIN) register, which is used to return status information as part of every response message sent by a slave device.

For reference, the IIN bit definitions are shown below:

| Bit | Description | Notes |
|---|---|---|
| 1 | Global (all stations) message received | |
| 2 | Class 1 data available | Set ON when the RTU generates one or more events of that particular class. Once all points of the class have been read by the master, the bit will be reset. |
| 3 | Class 2 data available | |
| 4 | Class 3 data available | |
| 5 | Time synchronisation required | Set ON when time synchronisation is required from the master. The bit is cleared when the master sets the time. |
| 6 | All digital outputs in local state | Set ON when some or all of the outstation's digital output points are NOT accessible by the DNP3 protocol |
| 7 | Device trouble | Set ON when an abnormal condition exists in the Outstation |
| 8 | Device restart | |
| 9 | Function code not implemented | Set ON if the requested function code is not implemented |
| 10 | Requested object unknown | Set ON If the requested object is not found by the driver |
| 11 | Parameter invalid or out of range | Set ON if a requested parameter is invalid or out of range |
| 12 | Event buffers have overflowed | Set ON if the oldest un-read event has been overwritten |
| 13 | Request understood but already executing | |
| 14 | Configuration corrupt | |
| 15 | Reserved | |
| 16 | Reserved | |

### 13.6.7 Further Information:

| | |
|---|---|
| **Support website:**<br>(registration required) | http://helpdesk.cse-semaphore.com |
| Contact us directly: | http://www.cse-semaphore.com/support/ |